UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB
CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT
CURSO DE MESTRADO EM INFORMÁTICA

RECONSTRUÇÃO E VISUALIZAÇÃO DE OBJETOS 3-D

Zairton Bastos Pinheiro

CAMPINA GRANDE - PB

Julho - 1993

RECONSTRUÇÃO E VISUALIZAÇÃO DE OBJETOS 3-D

Zairton Bastos Pinheiro

Dissertação apresentada ao curso de MESTRADO EM INFORMÁTICA da UNIVERSIDADE FEDERAL DA PARAÍBA, em cumprimento às exigências para obtenção do grau de mestre.

Área de Concentração : Ciência da Computação

PAULO ROBERTO CAMPOS DE ARAÚJO
Orientador

Campina Grande - PB

Julho - 1993

P654r Pinheiro, Zairton Bartos
Reconstrucao e visualizacao de objetos 3-D / Zairton Bartos Pinheiro. - Campina Grande, 1993.
148 f. : il.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Computacao Grafica 2. Sistema de Modelagem 3. Algoritmo de Modelagem 4. Dissertacao I. Araujo, Paulo Renato Campos de, Dr. II. Título

CDU 004.92(043)

RECONSTRUÇÃO E VISUALIZAÇÃO DE OBJETOS 3-D

ZAIRTON BASTOS PINHEIRO

DISSERTAÇÃO APROVADA EM 14.07.1993

PAULO ROBERTO CAMPOS DE ARAÚJO, M.Sc.
Orientador

MARIA DE FÁTIMA Q.V. TURNELL, Ph.D
Componente da Banca

GUILHERME VILAR, Dr.
Componente da Banca

MÁRIO TOSCANO DE BRITO FILHO, Dr.
Componente da Banca

Campina Grande, 14 de julho de 1993

AGRADECIMENTOS

Quando um trabalho se alonga muito (em termos de tempo), certamente muitas pessoas acabam por se envolver e colaborar com o mesmo, em alguma de suas fases, tornando difícil a inclusão de todas elas em uma página de agradecimentos como esta. Assim, gostaria inicialmente de agradecer a todas as pessoas que de alguma forma participaram deste trabalho, e em especial a :

Arthur Costa (Chefe da Divisão de Editoração Eletrónica)
Chen Wen Lin (Técnico do TCU)
Calson (Técnico do CNPq)
Duílio Torres Filho (Técnico do DENEMET/ONU)
Haroldo Castro Filho (Professor da UFMA)
Hélio Kuramoto (Chefe do Departamento de Informática)
Homero Piccolo (Professor da UNB)
Jaime Leiro (Técnico do IBICT)
Jairo da Silva (Gerente da Ciência Informática)
Julio Lopes (Consultor em Informática)
Luiz Antônio Garcia (Técnico do TST)
Marcelo Ladeira (Professor da UNB)
Mauro Kenjy (Técnico do IBICT)
Paulo César Zeredo (Técnico da TABRA Informática)
Pereira (Técnico do CNPq)
Silvia Barcellos (Técnica do IBICT)

pela colaboração em diversos momentos deste trabalho.

À minha família pela ajuda, apoio e compreensão.

Ao orientador deste trabalho, Prof. Paulo Roberto Campos de Araújo, pela sugestão do tema de dissertação, fornecimento de informações, estímulo e a sempre pronta disposição em colaborar, sem a qual este trabalho não teria sido realizado.

Ao ensino público e gratuito do Brasil, que torna possível a realização de trabalhos como este.

ÍNDICE ANALÍTICO

Página

LISTA DE ILUSTRAÇÕES

Figura Página

RESUMO

Este é um sistema de modelagem capaz de produzir um modelo tridimensional (baseado em octree), a partir de uma seqüência de imagens de seções de um objeto.

O sistema originalmente foi projetado para trabalhar com imagens de tomografos. Sua principal contribuição esta na elaboração de algoritmos de modelagem e síntese de imagens para objetos **reais** em uma plataforma de pequeno porte como um IBM/PC - 80386.

ABSTRACT

This is a modeling system capable of producing a tridimentional model (based on octree), using image sequences of sections of are object.

Originally, the system was developed for tomograph images. Its main contribuition is to create algorithms for modeling and image synthesis of **real** objects in a small platform such as an IBM/PC - 80386.

CAPÍTULO 1

INTRODUTÓRIO

1.1) Introdução

A Ciência moderna não pode ser expressa somente através de palavras, pois sequências de DNA, modelos moleculares, imagens médicas, mapas cerebrais, simulação de fluxo, etc..., todos necessitam de comunicação visual.

Devido à tecnologia agora disponível para coleta de dados (satélites em órbita da terra, sondas espaciais, supercomputadores, radiotelescópio, sondas geológicas e equipamentos médicos como os tomógrafos) muitas áreas de pesquisa estão se deparando com o problema de analisar e interpretar grandes volumes de dados.

O trabalho de análise e interpretação destes dados não é trivial, e se faz necessário o uso de recursos gráficos que condensem e agrupem a informação, dando-lhe uma forma visual, isto porque, utilizando exclusivamente formatos numéricos, o cérebro humano, que possui cerca de 50% de seus neurônios associados a visão, não é capaz de interpretar gigabytes de dados.

Podemos perceber que existe um conjunto de áreas de pesquisa, que passam pelo mesmo tipo de dificuldade (analisar e interpretar grandes volumes de dados) e em geral as técnicas desenvolvidas para cada uma delas são específicas, por dois motivos básicos :

a) a capacidade de representação dos modelos adotados, não é suficientemente robusta para representar, de forma simples, objetos de áreas diferentes, atendendo aos requisitos necessários.

b) os algoritmos que tratam cada modelo, são específicos para os mesmos e necessitam de um grande número de cálculos para cada aplicação, inviabilizando a adaptação dos mesmos para outras áreas.

Especificamente para a área médica, podemos perceber que a habilidade de um médico, em diagnosticar e tratar alguns tipos de doenças ortopédicas ou morfológicas (fraturas, abcessos, neoplasmas, inflamações, etc...), está intimamente ligada à visão.

A visualização científica, aplicada às imagens médicas, tem criado oportunidades de :

a) melhoria na qualidade dos diagnósticos;

b) projetos de planos de cirurgias;

c) detecção de lesões no cérebro;

d) projeto de próteses ortopédicas;

e) planos de tratamento com radiação.

Tudo isto através de imagens de porções do corpo, mostrando regiões inacessíveis através de visão direta.

Quando pensamos em radiologia, percebemos que a visualização constitui a sua base, pois os seus diagnósticos são obtidos a partir da análise de imagens radiográficas (onde o objeto radiografado é projetado em um plano) ou tomográficas (onde cada

tomograma mostra uma seção do objeto tomografado). O método mais comum de análise destas imagens é por observação visual.

Para o caso da radiografia, o especialista deve mentalmente imaginar o objeto radiografado a partir de sua projeção plana. Na radiografia tradicional, várias estruturas anatômicas encontram-se superpostas e apresentando distorções geométricas de dilatação.

A tomografia não apresenta os problemas da radiografia, no entanto, as informações, a respeito do formato e relacionamento espacial das estruturas, são difíceis de serem obtidas a partir de cortes do objeto. O relacionamento dos cortes do objeto é feito mentalmente por um radiologista que elabora um laudo médico a respeito das imagens. Caso esta reconstrução tridimensional possa ser realizada por computador, os médicos não radiologistas terão maior facilidade em observar e diagnosticar uma patologia.

1.2) Objetivos

Este trabalho de dissertação de mestrado, se propõe a atingir os seguintes objetivos :

a) Desenvolvimento de uma ferramenta para síntese de imagens de objetos modelados através de OCTREES, que possa ser diretamente utilizada na reconstrução e visualização do crânio humano ou de outros objetos que tenham sido digitalizados em um formato previamente estabelecido.

b) Análise comparativa de técnicas de iluminação e projeção de modelos OCTREES;

c) Lançar bases para outros trabalhos em síntese de imagens, que utilizem modelagem OCTREE; como por exemplo :

obtenção de uma imagem onde um conjunto de tecidos (cada um com um grau de transparência associado) pode ser visualizado.

d) Criar uma base de comparação entre a síntese de imagens utilizando modelagem OCTREE e outras técnicas de modelagem e síntese de imagens.

1.3) Área de concentração

. Visualização

Visualização é um conjunto de métodos computacionais, que transforma dados simbólicos em informação geométrica, habilitando o pesquisador a observar o resultado de uma coleta de dados, processamento ou simulação; ela permite ver o que aparentemente não é visível, enriquecendo o processo científico e em alguns casos criando um caminho revolucionário para a Ciência. A visualização abrange a compreensão e síntese de imagens, sendo então uma ferramenta para interpretar e gerar imagens.

1.4) Metodologia de trabalho

Para realizar a reconstrução tridimensional de objetos, em um computador, propomos basicamente seguir os seguintes passos :

a) Obtenção de imagens de cortes do objeto na forma digitalizada, o que permitirá o tratamento destas imagens por um sistema digital, que neste caso é um computador;

b) Pré-processamento destas imagens digitalizadas para a separação da porção da imagem que se deseja reconstruir;

c) Modelagem numérica e Visualização :

c.1) Geração do modelo de representação - esta é a forma como os dados estão representados e armazenados internamente;

c.2) Transformação geométrica/Projeção do modelo - consiste na obtenção da imagem do modelo quando observado em determinada posição;

c.3) Visualização do modelo em uma tela gráfica, utilizando técnicas de iluminação e síntese de imagens realistas apropriadas.

d) Possibilidade do usuário realizar uma operação de recorte no objeto, caso seja importante a visualização de alguma estrutura interna não visível;

Neste trabalho, o método de modelagem adotado é a OCTREE, que é caracterizada como uma técnica de decomposição celular do espaço do objeto (universo). Na modelagem OCTREE, o universo é subdividido em oito células de mesmo tamanho, se qualquer uma das células resultantes é homogênea, isto significa que ela está totalmente dentro (cheia) ou totalmente fora (vazia) do objeto, e não se realiza nenhuma subdivisão para ela. Por outro lado, se a célula é heterogênea, isto é, intercepta a fronteira do objeto, ela é novamente subdividida. O processo de subdivisão pára quando todas as células são homogêneas para uma dada resolução.

A modelagem via OCTREE, por ela mesma, já realiza uma economia de memória. Se adotarmos uma forma de codificação da OCTREE, onde atribuímos um código somente para as células cheias, teremos uma economia de memória ainda maior e a OCTREE será denominada LINEAR.

Os passos abaixo descritos foram realizados para concretizar o trabalho aqui proposto :

a) Levantamento bibliográfico;

b) Revisão da teoria básica de computação gráfica, que servirá como suporte para o desenvolvimento do trabalho;

c) Estudo da modelagem via OCTREE, bem como busca de uma melhor implementação para o modelo, de acordo com os recursos disponíveis;

d) Geração de uma OCTREE a partir de objetos matemáticos para teste da modelagem e que servirá também como massa de teste para as fases posteriores;

e) Estudo, análise e implementação de algoritmos para realizar operações de projeção e iluminação de objetos modelados através de OCTREES;

f) Estudo e implementação da aquisição de dados, que consiste no processo de digitalização dos tomogramas e classificação dos mesmos para separação dos tecidos relevantes para o trabalho (neste caso o ósseo).

g) Aquisição de dados reais, com o objetivo de testar a validade da ferramenta;

h) Elaboração do documento final da dissertação;

1.6) Descrição sumária da estrutura de capítulos

Os próximos capítulos deste trabalho foram estruturados conforme se segue :

No capítulo 2 foi feita uma descrição geral do sistema, para permitir ao leitor obter uma certa familiaridade com a maneira utilizada para modelar e produzir imagens, bem como permitir ao usuário poder operar o sistema.

No capítulo 3 procuramos informar ao leitor, qual o tipo e formato de dados que o sistema é capaz de reconhecer, bem como explicar o tipo de pré-processamento que o sistema realiza sobre estes dados para, em uma fase posterior, produzir o modelo.

No capítulo 4 procuramos explicar a teoria da modelagem via octree, bem como a forma como o modelo foi implementado neste trabalho e os motivos que nos levaram a adotar esta modelagem.

No capítulo 5 explicamos porque é necessário projetar o objeto e quais são os principais tipos de projeção que encontramos descritos na bibliografia, bem como a descrição de dois algoritmos utilizados neste trabalho para realizar as projeções.

No capítulo 6 procuramos fundamentar os conceitos de iluminação, bem como caracterizar os pontos mais importantes deste tipo de tratamento neste trabalho.

No capítulo 7 descrevemos as técnicas de (recorte) que foram incorporadas ao trabalho para permitir ao usuário visualizar estruturas internas ao modelo, que de outra forma não seriam visíveis.

Para finalizar no capítulo 8 apresentamos as conclusões do trabalho.

CAPÍTULO 2

DESCRIÇÃO DO SISTEMA

2.1) Introdução

Este capítulo foi escrito com o objetivo de promover uma visão geral da estruturação e funcionamento do sistema, a nível de usuário; facilitando assim a leitura e compreensão dos capítulos seguintes.

2.2) Características gerais do sistema

O sistema foi implementado em linguagem "C", e conta com uma interface gráfica orientada a menu e "help" sensível ao contexto para facilitar a sua utilização.

Neste trabalho utilizamos quatro tipos de "janelas". Os dois primeiros tipos funcionam em conjunto para montar um "menu" do tipo "sliding bar", que permite ao usuário o acesso a um dos quatro grandes módulos do sistema, bem como a seleção de uma função específica dentro do módulo (Figura 2. 1). O terceiro tipo funciona como uma janela para entrada de dados do tipo template[1] que é acionada a partir do momento em que o usuário realiza uma seleção (Figura 2. 2). O quarto tipo são janelas de mensagens que podem

---

[1]Todas as janelas do tipo "template" possuem os seus campos previamente preenchidos com valores "default" do sistema ou com valores digitados pelo usuário.

ser mostradas devido a ocorrência de algumas condições de exceção ou à solicitação de ajuda do usuário.

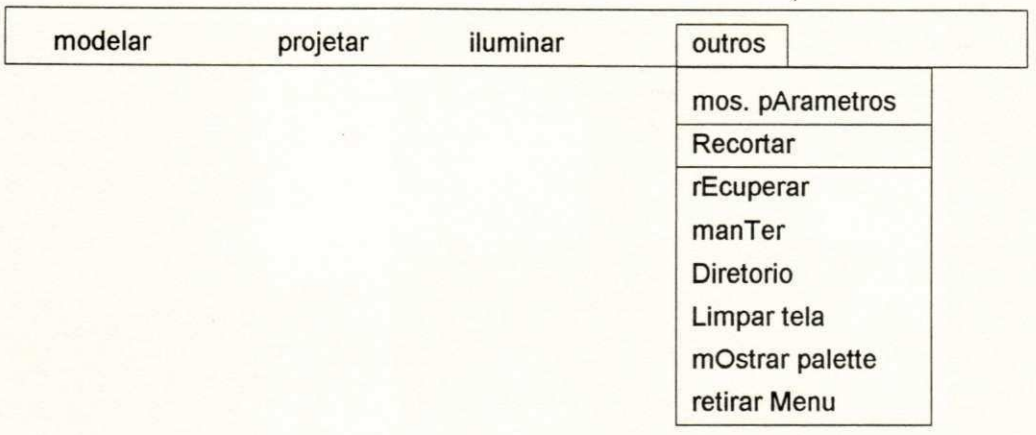


Figura 2. 1 - Menu principal do sistema

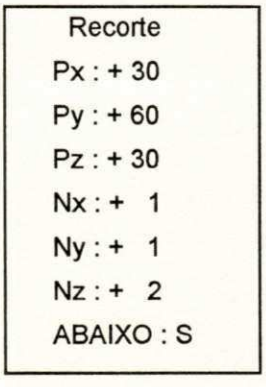


Figura 2. 2 - Janela do tipo template

Para que se possa utilizar o sistema devemos dispor de uma configuração mínima de hardware que inclui :

a) micro computador IBM PC/AT 80386 com coprocessador e sistema MS/DOS 5.00;

b) vídeo VGA ou super VGA;

c) 4 Mb de memória.

2.3) Conhecimento básico

Para que se possa operar o sistema é necessário o conhecimento de algumas de suas teclas de função do sistema; são elas :

a) Esc - esta tecla é utilizada para fechar qualquer janela aberta pelo usuário, bem como encerrar a execução do sistema[2];

b) Enter - esta tecla é utilizada para realizar uma seleção a nível do "menu" ou para deslocar o cursor para o próximo campo dentro de uma janela de entrada de dados;

c) Setas de direção - servem para deslocar o cursor dentro de um "menu" ou de uma janela para entrada de dados;

d) "PgDn" - esta tecla é utilizada dentro de uma janela do tipo "template" para validar e realizar a função selecionada;

e) Teclas Back Space, Insert e Delete - funcionam como convencionado na maioria dos editores.

f) "F1" - esta é a tecla de ajuda ao usuário.

2.4) Descrição dos módulos

O sistema encontra-se dividido em quatro grandes módulos, de acordo com a função a ser realizada. A cada módulo corresponde uma janela que permite ao usuário selecionar uma função dentro deste módulo.

---

[2]Em uma janela do tipo "template", os valores informados pelo usuário apenas permanecem digitados nos campos da tela, não ficam ativos para o sistema.

2.4.1) Modelagem

Este módulo do sistema é responsável por ler informações sobre cortes de um objeto e promover a sua reconstrução 3D através de um modelo octree.

Como ferramentas auxiliares para este trabalho de reconstrução, estão à disposição do usuário funções para gravar ou ler uma octree produzida pelo sistema, e também uma função para mostrar uma octree que esteja armazenada na memória do sistema (Figura 2. 3).

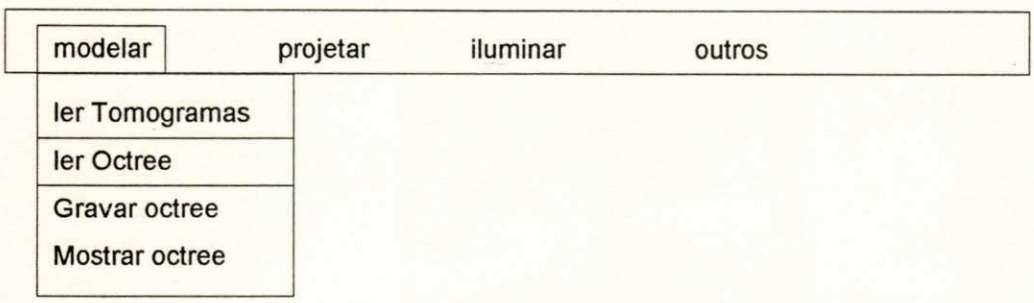


Figura 2. 3 - Menu de modelagem

Opções do módulo :

2.4.1.1) Ler tomogramas

Esta função é responsável pela leitura de uma seqüência de imagens de cortes de um objeto, em um formato previamente estabelecido, e pela produção de um modelo octree, capaz de reconstruir o objeto original a partir destes cortes.

Para que esta reconstrução possa ser realizada, as imagens devem estar no formato "PCX"[3], e o usuário deverá informar ao sistema sobre algumas características das imagens; são elas :

---

[3]Vide ítem 3.2

a) Resolução - todas as imagens devem possuir a mesma quantidade de pixels nas direções "x" e "y". Esta quantidade deverá ser informada ao sistema;

b) Quantidade de cortes - neste campo o usuário informa ao sistema o número de imagens que serão utilizadas na reconstrução.

c) Nome raiz - cada imagem utilizada na reconstrução deverá ter sido gravada em um arquivo cujo nome deve seguir uma regra de nomenclatura.

Os nomes dos arquivos devem começar por um prefixo padrão com até 5 letras chamado de nome raiz. Este deverá ser seguido de um número que identifique a imagem propriamente dita (Figura 2. 4).

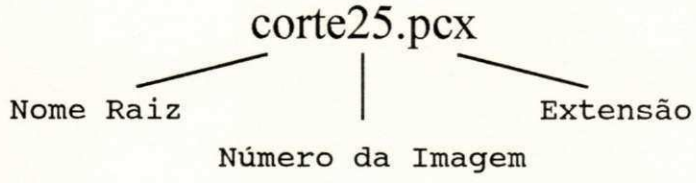


Figura 2. 4 - Regra de nomenclatura para as imagens dos cortes

d) Faixa de cinza - uma imagem pode ser composta de um grupo variado de tons de cinza (até 256 tons), de acordo com os elementos presentes na mesma.

O usuário deverá informar ao sistema em que limite de cinza está o elemento que deverá ser modelado (Figura 2. 5).

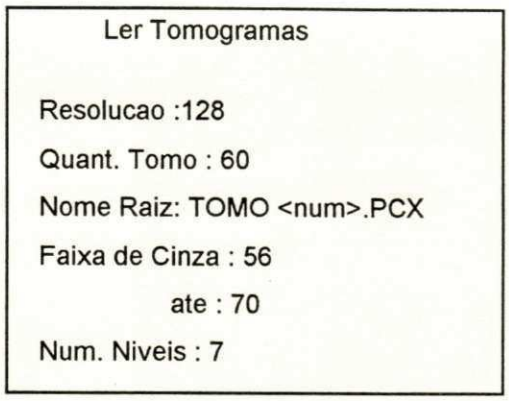


Figura 2. 5 - Janela da função ler tomogramas

2.4.1.2) Ler octree

Esta função permite ao usuário ler uma octree previamente preparada pelo sistema e armazenada em um arquivo com a extensão **oct**[4], carregando a mesma para a memória do sistema.

2.4.1.3) Gravar octree

Através desta opção é permitido ao usuário gravar, em um arquivo com a terminação "oct", a octree que se encontre carregada na memória do sistema.

2.4.1.4) Mostrar octree

Com esta função o usuário pode realizar uma rápida projeção da octree carregada na memória do sistema, com o objetivo de conferir se realmente esta é a octree a ser trabalhada.

2.4.2) Projeção

Este módulo do sistema é responsável por realizar a projeção do modelo, o que permite a sua visualização em tela. A tarefa de

---

[4]O usuário informa o nome do arquivo sem a extensão.

produzir uma projeção do modelo pode ser considerada como o primeiro passo no processo de geração da imagem.

As funções de projeção procuram sempre projetar todo o objeto, centralizando esta projeção na tela do monitor (Figura 2. 6).

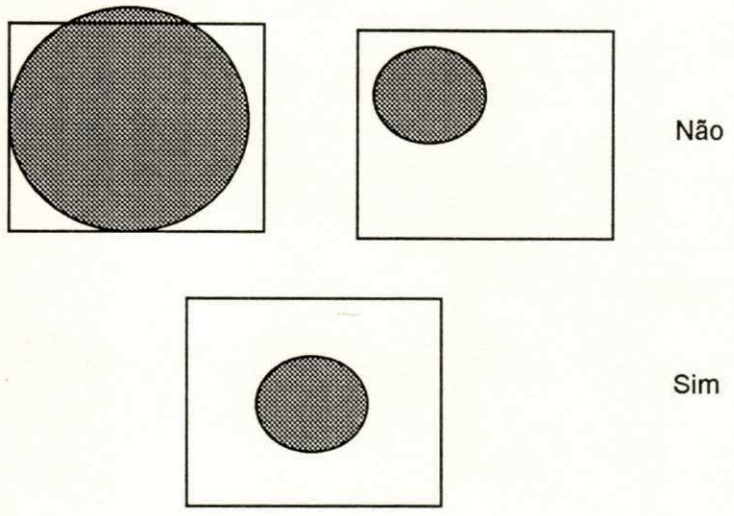


Figura 2. 6 - Forma de disposição de uma imagem na tela

A projeção realizada é do tipo ortogonal. Durante a projeção é mostrada, passo a passo na tela, uma imagem da projeção que está sendo construída.

Para realizar a projeção é necessário que o usuário do sistema informe uma direção de projeção no espaço através das informações de azimute e elevação.

Opções do módulo :

2.4.2.1) Projetar a octree

Esta função é responsável pela realização da projeção do modelo octree através de um algoritmo de "ray tracing".

Como os algoritmos de "ray tracing" são geralmente muito demorados, temos a opção de informar ao programa uma resolução para a projeção através dos parâmetros "tot. Pontx" e "tot. Ponty" (Figura 2. 7).

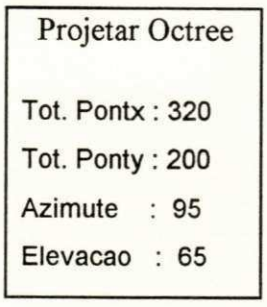


Figura 2. 7 - Janela da função projetar octree

2.4.2.2) Ler projeção

Internamente o sistema armazena a projeção em uma estrutura de dados do tipo z buffer.

Esta função é capaz de carregar, para a memória do sistema, o conteúdo de um z buffer, previamente gravado pelo sistema em um arquivo com extensão "buf".

2.4.2.3) Gravar projeção

Esta função permite ao usuário armazenar esta estrutura em um arquivo tipo "buf" para uma posterior recuperação.

2.4.2.4) Exibir projeção

Através desta função é permitido ao usuário visualizar o conteúdo do z buffer do sistema, por meio de uma imagem que é produzida a partir dele.

2.4.2.5) Rascunho

Esta função realiza a projeção do objeto modelado, utilizando um algoritmo de subdivisão recursiva.

Por este tipo de projeção ser extremamente rápido, não colocamos os parâmetros de resolução, só sendo informados os parâmetros que especificam a direção de projeção (Figura 2. 8).

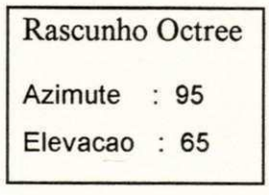


Figura 2. 8 - Janela da função de projeção rápida

2.4.3) Iluminar

Este é o segundo passo no processo de criação de uma imagem, após a projeção do objeto e remoção das superfícies ocultas. Para que tenhamos uma imagem de boa qualidade é necessário levar em conta a orientação, características da superfície, fontes de luz, e assim, calcular para cada ponto da superfície a sua intensidade de luz refletida.

Opções do módulo :

2.4.3.1) Cálculo do vetor gradiente

Para que possamos determinar a quantidade de luz refletida em um ponto de uma superfície, é necessário que determinemos a orientação da superfície neste ponto.

Existem muitos algoritmos utilizados para este cálculo [Chen85], [Heoh86], neste trabalho utilizamos dois algoritmos[5] que podem ser selecionados de forma independente, bastando para isto que o usuário escolha, no menu de iluminação, a opção "gradiente vizinho" ou "gradiente malha". Após escolher e informar ao sistema alguns parâmetros (se necessário), o sistema mostra ao usuário uma janela de mensagem informando que a função selecionada foi ativada.

2.4.3.2) Modelo de iluminação

Da mesma forma que o vetor gradiente, existem muitos modelos para calcular a intensidade de luz em um ponto da imagem [Chen85], [Hoeh87], [Gord85], [Fole82], [Hear86].

Neste trabalho utilizamos dois modelos: o primeiro deles considera somente a reflexão difusa da luz nas superfícies do objeto, e o segundo, considera tanto a reflexão difusa quanto a especular. Ambos os modelos adotam uma fonte de luz primária extensa de raios paralelos, associada à intensidade de luz ambiental.

Para selecionar um dos modelos de iluminação, devemos proceder do mesmo modo que fizemos para escolher o algoritmo de cálculo do vetor gradiente, só que escolhendo "iluminação difusa" ou "iluminação completa", informando os parâmetros adequados e teclando "PgDn". Assim, também veremos na tela uma mensagem que indica a ativação da função solicitada.

---

[5]Para maiores informações sobre estes algoritmos vide ítem 6.4.2

2.4.3.3) Função iluminar

Quando escolhemos esta função no menu de iluminação, o sistema aplica uma iluminação à projeção do objeto que esteja carregada em sua memória (z buffer), de acordo com as opções de "modelo de iluminação" e "cálculo do vetor gradiente", que se encontrem ativas.

2.4.3.4) Gravar a imagem

Através desta opção, o usuário pode gravar a imagem obtida na tela no formato "PCX".

Diferentemente da opção "gravar projeção" no menu de projeção (que guarda o z buffer em um arquivo), esta opção somente guarda uma imagem. Assim, esta opção funciona como uma forma de exportar uma imagem produzida pelo sistema para um outro software que seja capaz de entender o formato "PCX".

2.4.4) Outros

Neste módulo colocamos uma coletânea de funções para auxiliar o usuário nos processos de modelagem e geração de imagens.

Opções do módulo :

2.4.4.1) Mostrar parâmetros

Esta função abre uma janela que mostra o valor dos principais parâmetros do sistema.

2.4.4.2) Recorte

A função de recorte foi adicionada ao sistema para permitir ao usuário retirar partes do modelo, e assim observar melhor partes internas e externas do objeto.

Para que o recorte possa ser realizado, o usuário deverá definir para o sistema um plano de recorte do objeto, e qual dos subespaços definidos pelo plano deverá conter a parte do objeto que desejamos manter no sistema (Figura 2. 9).

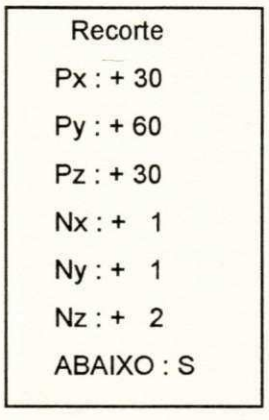


Figura 2. 9 - Janela da função de recorte

Para definir o plano de recorte, basta que o usuário informe para o sistema um ponto deste plano (Px, Py, Pz) e o vetor normal a este plano (Nx, Ny, Nz).

Cada uma destas coordenadas é informada em dois campos distintos. No primeiro campo o usuário informa para o sistema o sinal da coordenada. No segundo, o usuário informa o valor do módulo da coordenada.

Para efeito de orientação, consideramos o subespaço apontado pelo vetor normal informado como sendo o subespaço acima do plano de recorte (Figura 2. 10). Assim, para que o usuário informe

ao sistema qual o subespaço que deverá manter o porção do modelo que lhe interessa, ele deve responder sim (s) ou não (n) para indicar ao sistema se o subespaço que manterá a parte desejada do objeto, é ou não o subespaço abaixo do plano de recorte[6].

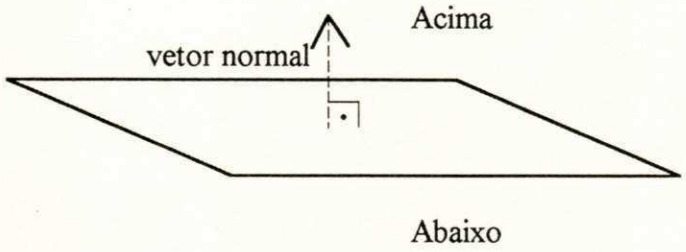


Figura 2. 10 - Orientação do plano de recorte

2.4.4.3) Manter

Dada a interatividade da função de recorte, esta sempre trabalha sobre o conteúdo original do modelo. Isto quer dizer que, se realizarmos muitas operações consecutivas de recorte, ao invés de obtermos um objeto com múltiplos cortes, obteremos o objeto original com o último recorte realizado. Isto permite ao usuário ajustar de forma interativa o plano de recorte.

Caso desejemos realizar múltiplos recortes sobre o modelo, devemos chamar a função "manter" que preserva o último recorte realizado.

2.4.4.4) Recuperar

Esta função permite ao usuário desfazer o último recorte realizado sobre o modelo, antes da aplicação de uma função manter.

---

[6]O subespaço abaixo é o que fica no sentido oposto ao apontado pelo vetor normal ao plano de recorte.

2.4.4.5) Diretório

Esta função tem o objetivo de mostrar ao usuário o conteúdo do diretório corrente, bem como permitir a mudança de diretório.

2.4.4.6) Limpar a tela

Esta função limpa a tela do sistema.

2.4.4.7) Mostrar "palette[7]"

Esta função mostra a distribuição de tons de cinza e cores na palette do sistema.

2.4.4.8) Retirar menu

Esta função esconde os menus do sistema que estejam na tela, permitindo ao usuário visualizar toda a tela, até que seja pressionada alguma tecla.

---

[7]Palette : tabela de cores utilizadas pelo sistema para "colorir" os seus objetos.

CAPÍTULO 3

LEITURA E CLASSIFICAÇÃO DE IMAGENS

3.1) Introdução

Como explicado no ítem 2.4.1 deste trabalho, o sistema aqui projetado é capaz de produzir um modelo tridimensional (3D) de um objeto, através de imagens de seções deste objeto (Figura 3. 1).

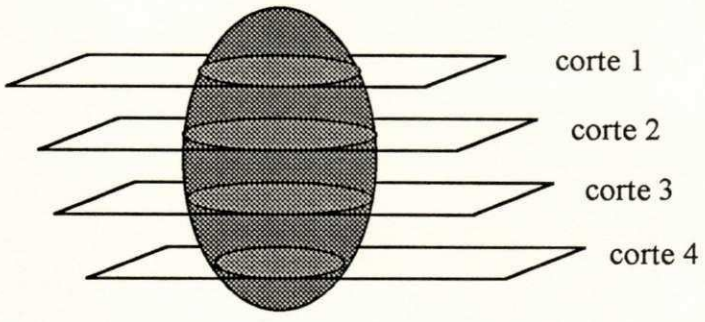


Figura 3. 1 - Objeto particionado

Para que o sistema possa reconhecer e ler estas imagens, elas devem seguir um padrão de codificação.

Existem muitos padrões utilizados para codificar imagens (TIF, PCX, EPS, BMP, RIFF, BRIM, ...).

Neste trabalho optamos pela utilização do padrão "PCX", dada a grande disponibilidade de softwares que são capazes de ler e produzir imagens codificadas segundo este padrão, na plataforma de hardware onde o sistema foi implementado.

O padrão de codificação "PCX" foi desenvolvido pela "Zsoft" para ser utilizado nos arquivos de seu software "PC Paintbrush". Dado o sucesso do software , o padrão "PCX" terminou transcendendo os limites do programa original, para se tornar um formato padrão para o intercâmbio de imagens no ambiente MS/DOS.

3.2) Padrão "PCX"

Todos os arquivos de imagens codificados segundo o padrão "PCX" possuem duas porções distintas : uma área de cabeçalho, onde são colocadas informações sobre a imagem, e uma área de dados, onde encontramos uma codificação da imagem propriamente dita.

3.2.1) Área de Cabeçalho

Esta é uma porção 128 bytes, colocada no início do arquivo, que possui informações de controle que possibilitam a leitura da imagem codificada no arquivo.

As informações de controle são :

a) identificação do arquivo - é um campo de um byte preenchido sempre com "0Ah", para identificar o arquivo como um arquivo do tipo "PCX".

b) versão do "PCX" - este campo de um byte contém a versão do programa que produziu o arquivo "PCX". Os valores possíveis são:

Valor	Versão do "PC Paintbrush"
00	2.5
02	2.8 sem palette
03	2.8 com palette
05	3.0

c) razão de bits/pixel - este campo de um byte contém a informação do número de bits, consecutivos do arquivo, utilizados para codificar um pixel da imagem. Os valores possíveis são :

bits/pixel	Modo
1	monocromático ou EGA/VGA com 16 cores;
2	CGA 4 cores;
8	MCGA/VGA 256 cores;

d) limites da imagem - são quatro campos de dois bytes que informam as dimensões da imagem (pcxXmin, pcxYmin, pcxXmax, pcxYmax). As dimensões da imagem serão :

pcxXmax - pcxXmin + 1 por pcxYmax - pcxYmin + 1

e) vídeo - são dois campos de dois bytes que guardam a resolução do modo de vídeo utilizado na criação deste arquivo (pcxHoris, pcxVert).

d) palette - é uma matriz de 16 x 3 campos de um byte que guarda a palette, para as imagens de 16 cores, no seguinte formato :

Red	Green	Blue	Cor
R1	G1	B1	*cor 1*
R2	G2	B2	*cor 2*
...	...	...	...
R16	G16	B16	*cor 16*

e) modo do vídeo - este campo de um byte contém o valor do modo de vídeo, na "BIOS" da controladora de vídeo utilizada.

f) planos de cores - este campo de um byte guarda o número de planos de cores utilizados para codificar esta imagem. Caso seja utilizado mais de um plano de cor, os dados da imagem aparecem como "scanlines" para cada plano. Ex:

scanline 1: 010111010100011010101010	plano 1
scanline 1: 101011010101010010101010	plano 2
scanline 1: 111111111111100011111101	plano 3
...	...
scanline n: 000000111111010111100101	plano 1
scanline n: 010101111111111100111100	plano 2
scanline n: 111100011111100110010110	plano 3

g) bytes / scanline - este campo de dois bytes informa o número de bytes que são usados para cada scanline. Este valor depende do número de bits por pixel e da dimensão x da imagem.

h) tipo de palette - este campo de dois bytes informa o tipo de palette utilizada na imagem:

1 - escala (tons) de cinza;
2 - colorida.

g) o restante dos 128 bytes (58 bytes) são reservados para futuras extensões do formato.

3.2.2) Área de dados

Os arquivos "PCX" utilizam uma codificação, em sua área de dados, chamada de "run lenght encoding" (RLE) [Horn87], para compressão de dados.

O algoritmo "RLE" procura aproveitar a repetição de um pixel com o mesmo atributo em uma scanline da imagem. A codificação consiste em colocar um byte com os dois bits mais significativos setados, para indicar que este byte contém informação que indica a quantidade de repetições do próximo byte na imagem.

A quantidade de repetições é armazenada nos 6 bits menos significativos do byte de repetição. Ex:

	repetição	dado	repetição	dado
hexa	FF	00	D1	00
binário	11|111111	00000000	11|010001	00000000

Quando é utilizada a codificação "RLE", e se deseja armazenar um dado que possui os dois bits mais significativos setados, deve-se inserir antes deste dado um byte de controle indicando uma repetição única. Ex:

	repetição	dado
hexa	FF	D6
binário	11|000001	11010110

3.2.2.1) Algoritmo de descompactação dos dados

```
Início
    Enquanto não terminar a área de dados faça :
    Início
        Ler um byte;
        Se os dois bits mais significativos estão "ligados" então:
        Início
            Guardar número de repetições, "N" (6 bits restantes);
            Ler o próximo byte;
            Repetir este byte "N" vezes;
        Fim
        Se não então:
            Este é um byte de dados;
    Fim
Fim
```

3.3) Classificação da imagem

Pelo que se pode perceber da codificação "PCX", existem muitos tipo de arquivos "PCX". O sistema é capaz de ler somente dois destes tipos, os monocromáticos (binários) ou os arquivos com 256 tons de cinza.

No caso das imagens monocromáticas, o sistema lê e considera para fins da modelagem, os pontos da imagem com bits "ligados".

No entanto, no caso das imagens com 256 tons de cinza, é permitido ao usuário especificar uma faixa de tons de cinza, onde se espera encontrar, nas imagens, o objeto a ser modelado.

Muitos erros podem ser produzidos a partir desta classificação simples, isto porque não se tinha, neste trabalho, o objetivo de tecer grandes considerações nesta matéria, que deve ser tratada em um

trabalho a parte. Neste trabalho simplesmente colocamos esta opção para tornar mais simples este processo de aquisição de dados.

Caso esta classificação não atenda às necessidades de quem por ventura venha a utilizar este trabalho, este processo de classificação pode ser suprimido simplesmente por fornecer ao sistema imagens já classificadas e armazenadas em arquivos monocromáticos (binarizados).

CAPÍTULO 4

MODELAGEM

4.1) Introdução

Modelos são utilizados para representar entidades físicas, abstratas ou fenômenos, não somente com o propósito de criar uma imagem, mas em geral, representar sua estrutura e seu comportamento [Fole82], facilitando a nossa compreensão a respeito da coisa modelada.

Os modelos podem ser divididos segundo [Fole82] em :

a) modelos organizacionais - que se preocupam em representar a organização de uma empresa ou a classificação de elementos como no caso da taxonomia;

b) modelos quantitativos - cujo objetivo principal é representar grandezas (econômicas, sociais, climáticas, físicas, etc..);

c) modelos geométricos - que estão preocupados em representar ou dar uma representação geométrica para determinadas características da entidade modelada. Estes modelos são muito utilizados na engenharia, arquitetura, química, medicina, etc... .

Segundo [Fole82], a palavra chave em modelagem é criar modelos que representem propriedades das entidades modeladas, utilizando algum tipo de formalismo.

Neste trabalho o interesse se volta exclusivamente para os modelos geométricos. Assim, de agora em diante, quando falarmos em modelo, estaremos nos referindo a modelos geométricos.

No processo de modelagem existem alguns critérios que devem ser perseguidos para que tenhamos uma boa modelagem.

REQUICHA [Requ80] nos dá uma lista de propriedades desejáveis para uma boa modelagem, são elas :

a) O domínio do método de modelagem deve ser grande o suficiente para permitir uma modelagem simples de um conjunto variado de objetos;

b) A modelagem não deve ser ambígua, ou seja, para um dado modelo deve corresponder um único objeto e vice-versa;

c) Acurácia que permita o objeto ser representado sem aproximação, ou pelo menos com um grau de precisão desejado;

d) A modelagem não deve permitir a geração de modelos inválidos, ela deve produzir modelos corretos por construção.

e) A modelagem deve ser tão simples que permita a automação do processo de modelar;

f) A modelagem deve ser compacta, para permitir uma economia de memória em sua implementação.

Na tentativa de atender a maioria destes objetivos, alguns sistemas implementam híbridos dos métodos clássicos de modelagem.

Foley [Fole82] nos diz que projetar uma técnica de modelagem que atenda a todos estes requisitos é um trabalho muito difícil, e certamente alguns critérios serão comprometidos a favor de outros, de acordo com os objetivos a serem atendidos.

4.2) Tipos de Modelagens Geométricas

Agora, vamos expor brevemente alguns tipos de modelagens geométricas, para situar melhor onde se classifica a modelagem por octree.

Podemos classificar a modelagem, em dois grandes grupos:

a) Modelagem Sólida - que trata de representação do volume do objeto através de primitivas 3D que permitem mapear não somente a superfície do objeto mas também o seu volume.

b) Modelagem de Superfície - que procura representar a superfície do objeto através de primitivas 2D.

A OCTREE é uma modelagem do tipo sólida e por isto vamos nos concentrar nos modelos sólidos.

Para modelagem sólida, podemos listar os seguintes modelos clássicos :

4.2.1) Primitive Instancing

Nesta modelagem, definimos um conjunto de primitivas sólidas (3D), relevantes para uma dada área de aplicação.

Estas primitivas são parametrizadas em termos de propriedades e operações que podem ser realizadas sobre as mesmas (rotação, translação, escala, etc...).

A partir das mudanças destes parâmetros, obtemos novas instâncias da primitiva e conseqüentemente novos objetos.

Para exemplificar, podemos ter uma primitiva que seria uma pirâmide (Figura 4. 1), parametrizada em termos de :

a) número de faces;

b) altura

c) raio da base

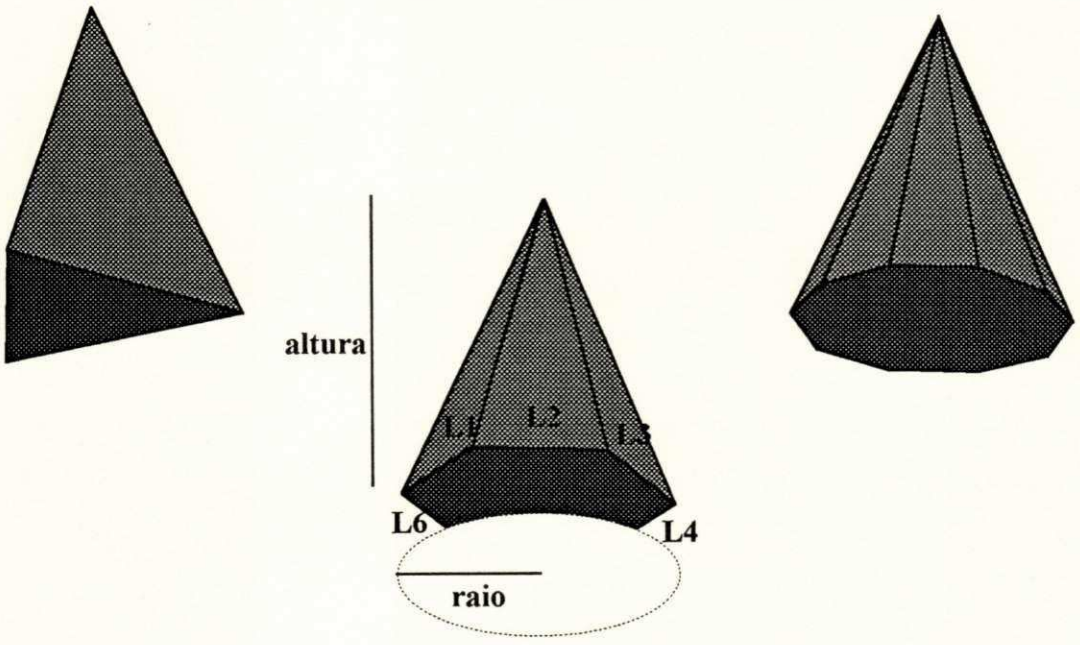


Figura 4. 1 - Primitive instancing

Este tipo de modelagem é utilizado para objetos complexos, que são tediosos para definir e dos quais já se possui um padrão bem definido.

4.2.2) Sweep Representation

O que caracteriza esta técnica de modelagem é a obtenção do objeto desejado a partir do arrasto de um outro objeto 2D ao longo de uma trajetória normal ao plano que o contém, obtendo assim um objeto 3D.

Outra forma de "sweep" consiste em rotacionar o objeto 2D em torno de um eixo, obtendo assim um volume de revolução (Figura 4. 2).

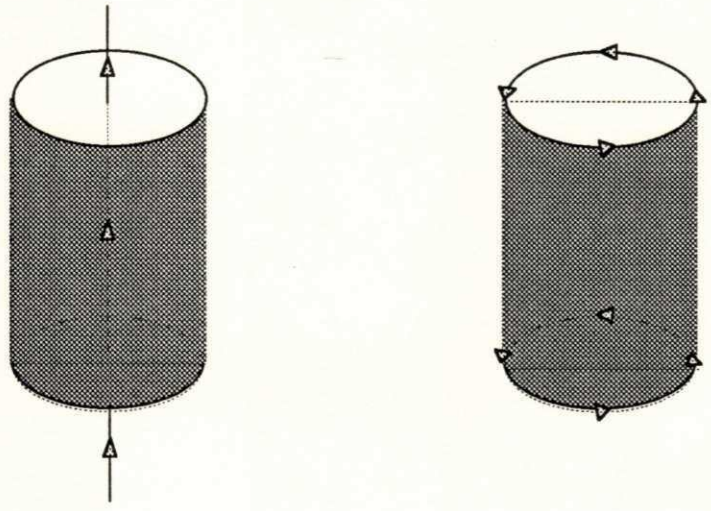


Figura 4. 2 - Sweep representation

4.2.3) Constructive Solid Geometry (CSG)

Nesta forma de modelagem existe um conjunto de operações booleanas (inclusas na representação) para produzir o modelo.

Assim, um objeto é modelado como uma árvore cujos nós são operações booleanas e cujas folhas são primitivas da representação.

A partir da combinação das primitivas, seguindo a hierarquia das operações booleanas, obtemos o objeto (Figura 4. 3).

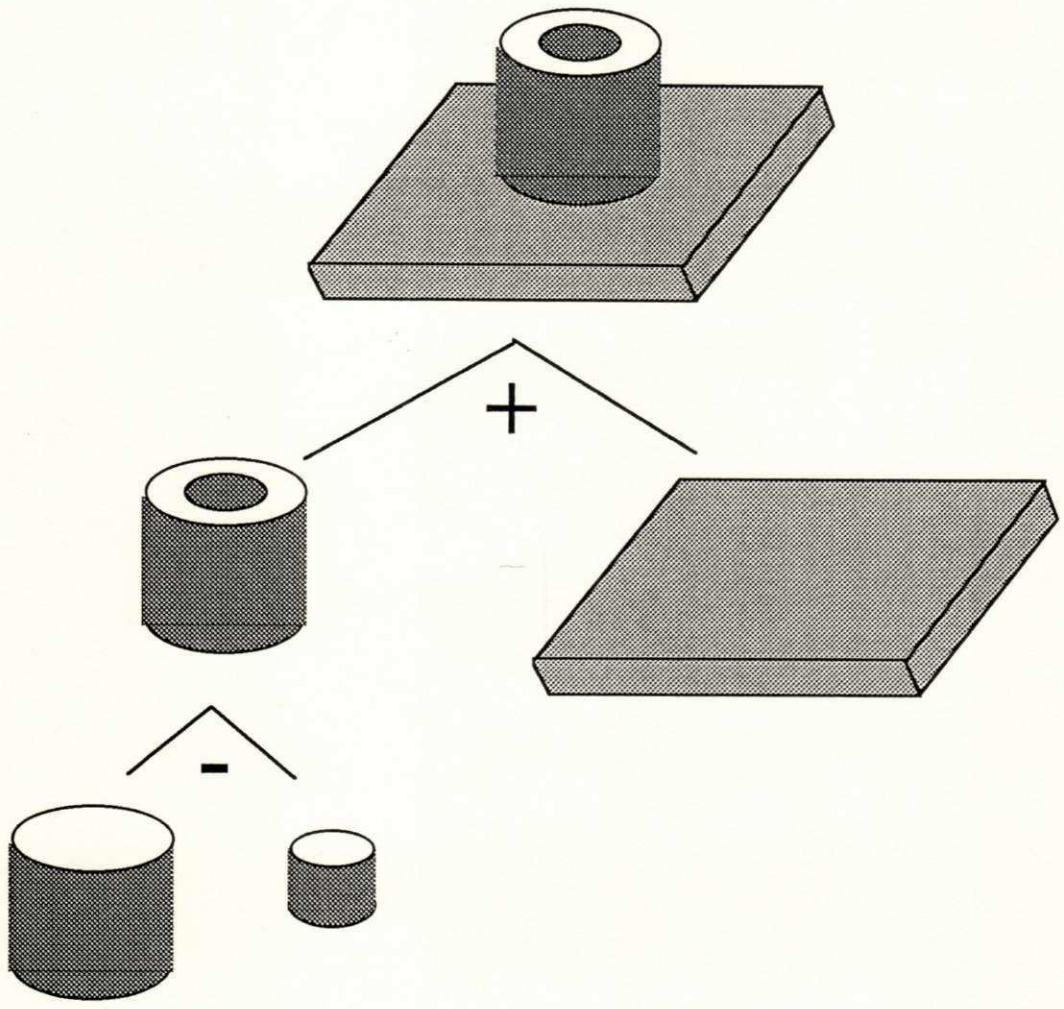


Figura 4. 3 - Modelagem CSG

Com relação a este tipo de modelagem podemos dizer que :

a) os algoritmos para combinar as primitivas e realizar as operações booleanas são de difícil implementação;

b) escolher um conjunto de primitivas adequadas para o domínio da modelagem é uma tarefa difícil;

c) a modelagem CSG não possui um modelo único para um dado objeto (Figura 4. 4);

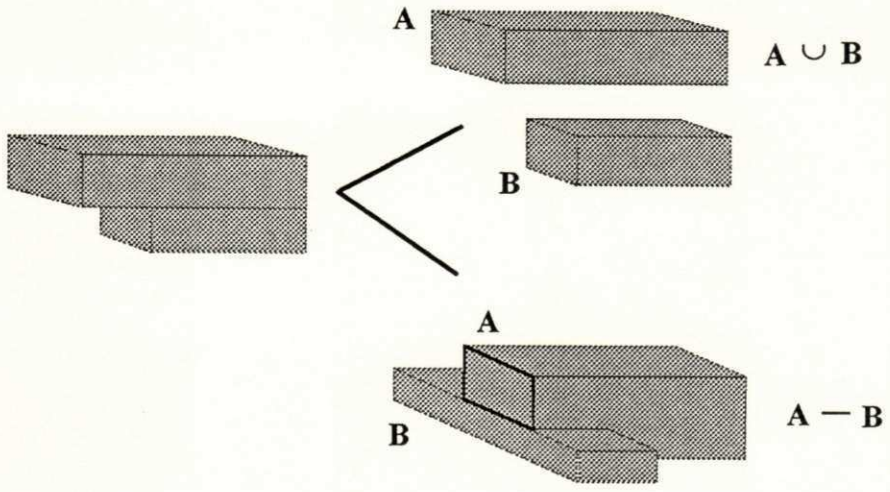


Figura 4. 4 - Dois modelos CGS para um único objeto

4.2.4) Spatial partitioning representation

Falaremos agora de uma classe de modelagem dentro da qual está inserida a modelagem por octrees.

Nesta classe, um objeto é modelado por sua decomposição em uma coleção de primitivas sólidas disjuntas (que não se interceptam).

Uma primitiva sólida pode variar em termos de tipo (cubos, cilindros, esferas), tamanho, posição e orientação no espaço.

As principais técnicas de modelagem que compõem esta classe[1] são: cell decomposition (decomposição celular), spatial occupancy enumeration (ocupação espacial por enumeração) e octrees.

Falaremos agora brevemente sobre cada uma das técnicas nos atendo mais à modelagem octree.

---

[1]Estas técnicas podem ser entendidas como casos especiais de CSG onde temos um único operador de adição.

4.2.4.1) Cell decomposition

Nesta técnica, temos o objeto modelado a partir de um conjunto de células primitivas. Cada célula não pode interceptar outras células e elas devem compartilhar entre si um ponto, uma reta ou uma face.

Esta técnica é considerada a mais geral da classe[Fole82] e não é ambígua, no entanto ela não é necessariamente única, pois um objeto pode ter dois modelos distintos, que o representem (Figura 4. 5), o que poderia trazer problemas para automatizar o processo de modelagem.

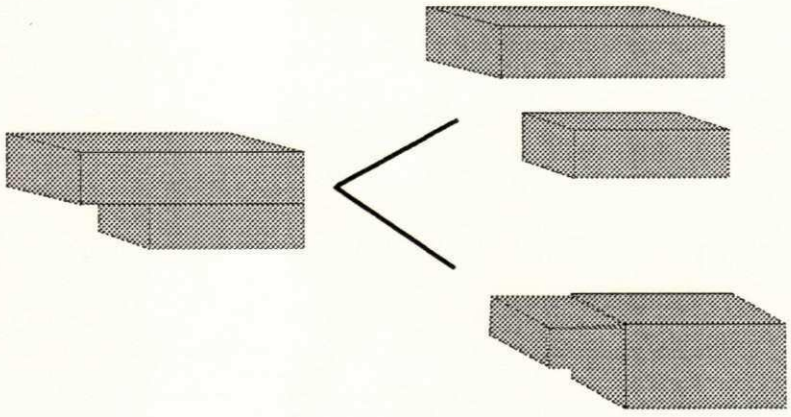


Figura 4. 5 - Cell decomposition

Outro ponto problemático é a validação de um modelo, para garantir que as células não se interceptem e ao mesmo tempo compartilhem um vértice, uma aresta ou uma face.

4.2.4.2) Spatial occupancy enumeration

Esta técnica pode ser olhada como um caso especial de cell decomposition, onde o objeto é decomposto em células idênticas, chamadas "voxels" (elementos de volume), dispostas em uma matriz (Figura 4. 6).

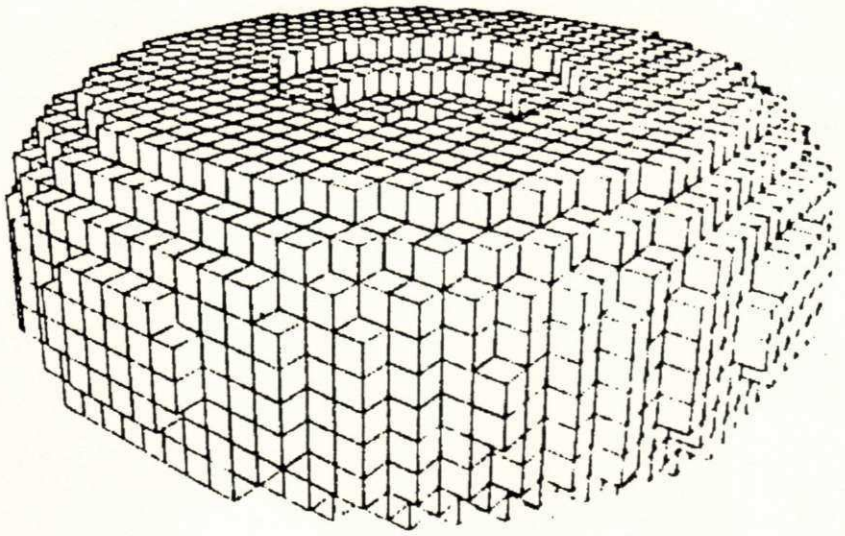


Figura 4. 6 - Modelagem cuberille

O formato da célula pode variar, sendo o cubo o mais comumente utilizado, dando origem ao nome da técnica de modelagem por um array de cubos ¨cuberille¨.

Neste tipo de modelagem, como já existe uma matriz predefinida, o objeto é modelado através do agrupamento destas células, de acordo com o objeto modelado. A vantagem desta modelagem é que ela não é ambígua e é única.

Por ter uma malha fixa de células, onde podemos observar o objeto particionado em camadas de células, esta modelagem é utilizada na área biomédica para representar objetos tomografados, onde cada camada da malha é atribuída a um tomograma (Figura 4. 7).

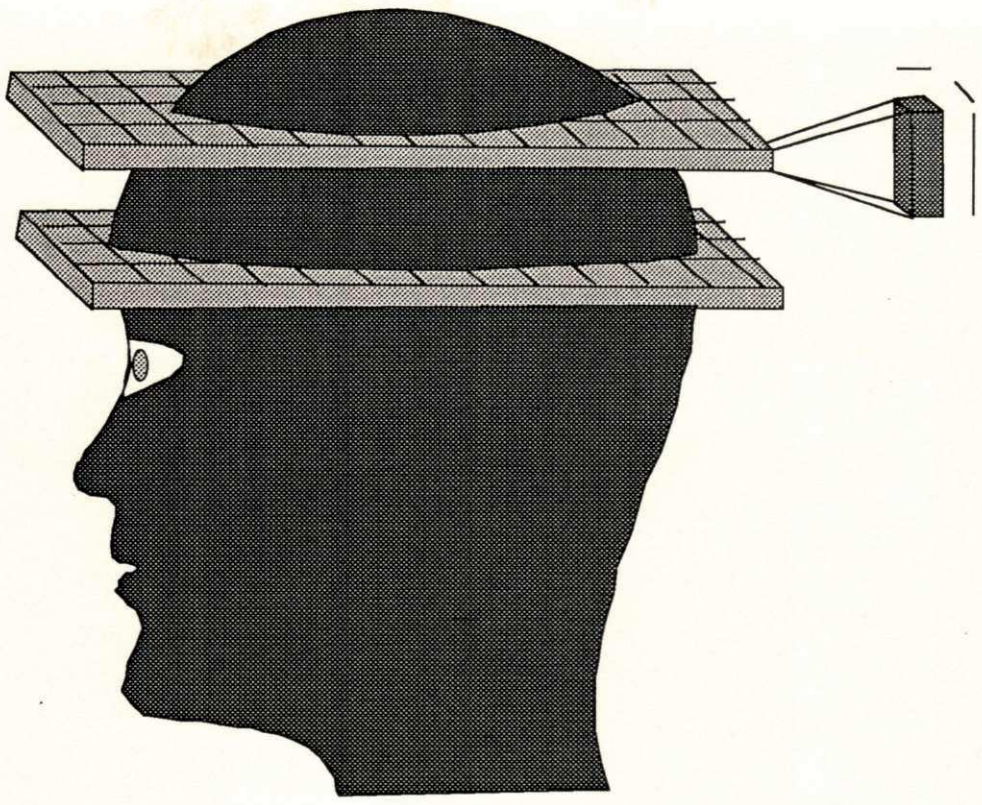


Figura 4. 7 - Objeto tomografado

Podemos destacar dois problemas com este tipo de modelagem :

a) As células possuem um formato fixo, sendo possível a obtenção de objetos somente por meio de aproximação (em geral).

b) Para obter um modelo mais preciso do objeto, devemos diminuir o tamanho das células na matriz, o que provoca um aumento no número de células e conseqüentemente o aumento da memória necessária para guardar o modelo.

4.2.4.3) Octree

4.2.4.3.1) O que é

O termo octree surgiu da junção do prefixo "oct" (que nos traz a idéia de oito) com a palavra "tree" (do inglês árvore), ou seja uma árvore onde cada nó possui oito filhos (Figura 4. 8).

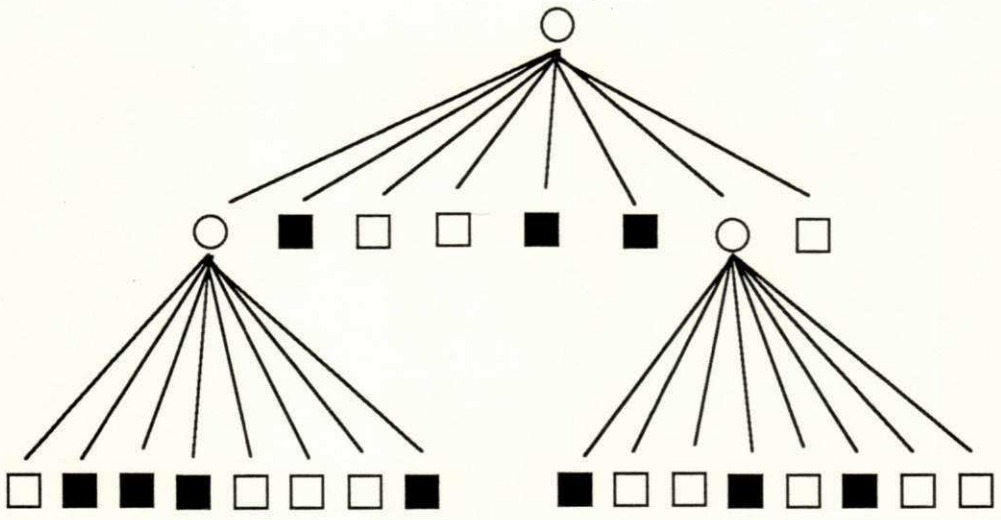


Figura 4. 8 - Representação gráfica de uma octree convencional

A modelagem por octree pode ser entendida como uma variante do método spatial occupancy enumeration, para resolver o problema de quantidade de memória necessária para armazenar o modelo, quando diminuimos o tamanho da célula.

4.2.4.3.2) Como surgiu

A modelagem octree toma como base a idéia de dividir para conquistar, e surgiu como uma generalização da técnica de

modelagem de imagens por quadtrees[2] como apresentado por [Meag82].

Vamos fazer uma breve exposição a respeito da modelagem de figuras (2D) com quadtrees, para podermos entender melhor o processo de geração de um modelo octree.

Pensemos na imagem de uma esfera (Figura 4. 9), a geração da quadtree para esta imagem começa por inicialmente definir um "grande quadrado" que seja capaz de conter toda a imagem, este será o nosso universo, de onde começaremos o processo de subdivisões sucessivas em busca de áreas de padrões homogêneos.

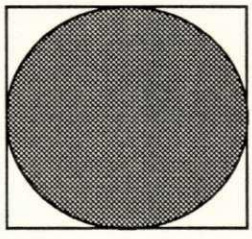


Figura 4. 9 - Primeiro passo na geração de uma quadtree

Definido o universo, que seria o nível 0 da quadtree, pesquisamos este nível para descobrir se este encontra-se totalmente homogêneo, em termos da característica que estamos analisando (Figura 4. 10). Caso a resposta seja positiva, nossa quadtree só terá o nível zero ou raiz. Caso a resposta seja negativa, o universo será subdividido por dois segmentos de reta em quatro quadrados de mesmo tamanho (Figura 4. 10).

---

[2]Quadtrees tem sido utilizadas por aplicações em processamento de imagens [Tani74] como forma de medir a complexidade de uma imagem.

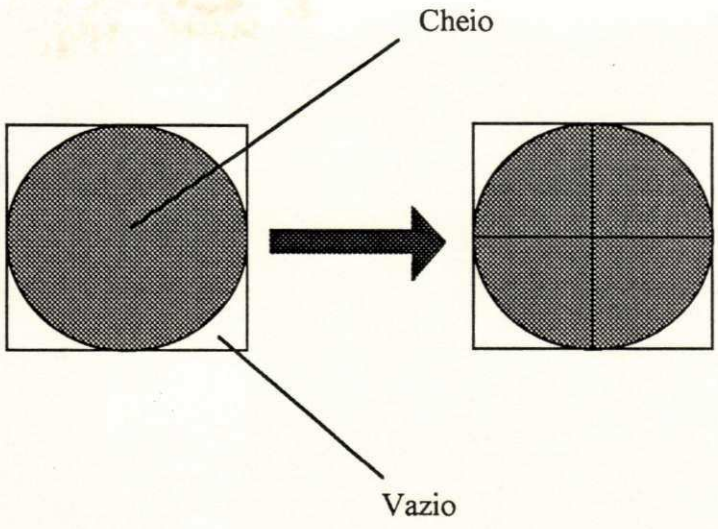


Figura 4. 10 - Segundo passo na geração de uma quadtree

Novamente analisamos cada um deste quadrados de nível 1, quanto a sua homogeneidade (Figura 4. 11).

Cada novo quadrado passará por um novo processo de subdivisão idêntico ao do pai, caso este ainda não seja considerado homogêneo (Figura 4. 11).

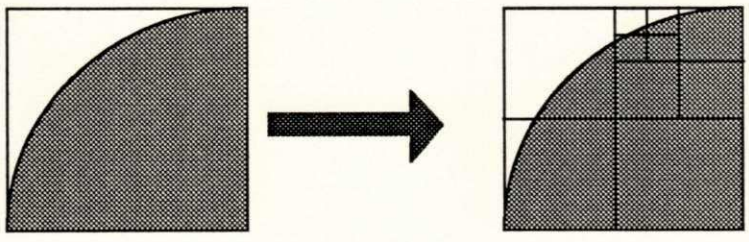


Figura 4. 11 - Terceiro passo na geração de uma quadtree

Este processo de subdivisões sucessivas termina quando o quadrado for considerado homogêneo ou tiver sido atingida a máxima resolução definida para o modelo.

Podemos notar que a quadtree é um modelo hierárquico, onde quadrados de um nível "n" são subdivididos em 4 quadrados de um nível "n+1".

Este modelo hierárquico pode ser representando e armazenado em uma estrutura de árvore (Figura 4. 12) ou em uma estrutura de lista linear como veremos mais tarde quando falarmos de octree linear.

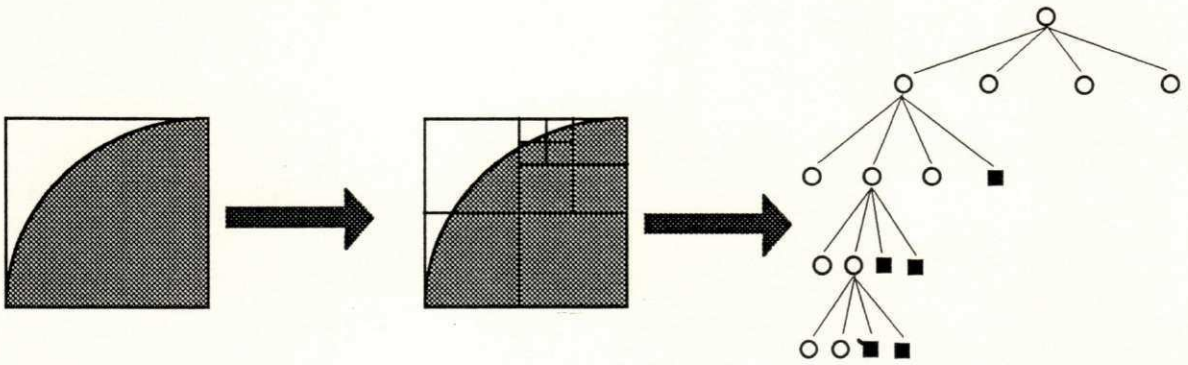


Figura 4. 12 - Representação em árvore para uma quadtree

A forma de modelar uma imagem (2D) por uma quadtree, foi extendida para um objeto (3D), originando a técnica de modelagem octree (como citado anteriormente).

A extensão consiste em se considerar o universo não mais como um quadrado no plano, mas sim como um cubo no espaço (Figura 4. 13), que será subdividido por conjuntos de 3 planos, em 8 células cúbicas de mesmo tamanho (Figura 4. 14), caso não seja considerado homogêneo.

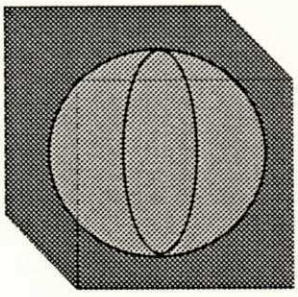


Figura 4. 13 - Extensão da modelagem quadtree

Vejamos agora como fica o nosso caso da esfera, não mais trabalhando com a imagem da esfera, mas com o objeto esfera em 3D (Figura 4. 14).

Subdividimos o universo (nível 0) por três planos que:

a) São perpendiculares entre si;

b) São paralelos a uma das faces do universo;

c) Passam pelo centro do universo;

Isto produz 8 células (octantes) de mesmo tamanho (Figura 4. 14);

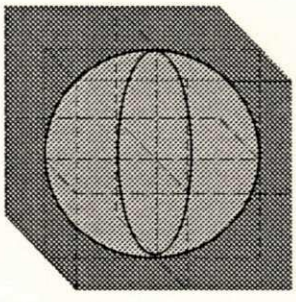


Figura 4. 14 - Primeiro passo da modelagem por octrees

Caso qualquer destas células (octantes) resultantes seja homogênea, isto significa que :

a) Ela está totalmente dentro do objeto (cheia);

b) Ela está totalmente fora do objeto (vazia).

Assim, a subdivisão para este nó não acontece (Figura 4. 15).

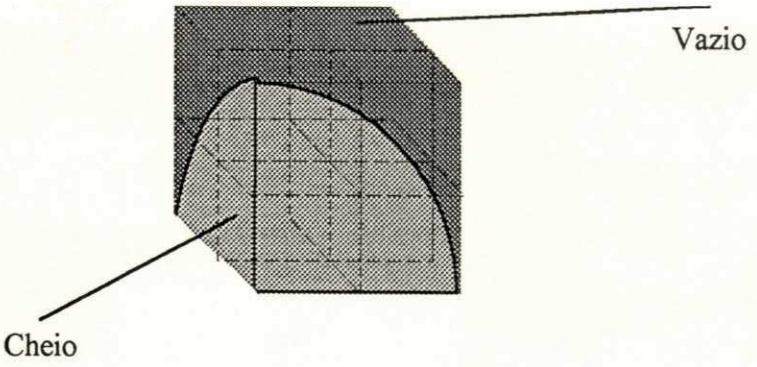


Figura 4. 15 - Segundo passo da modelagem por octrees

Por outro lado, se a célula é heterogênea, isto é, intercepta o objeto, ela é subdividida da mesma forma que o universo (Figura 4. 16).

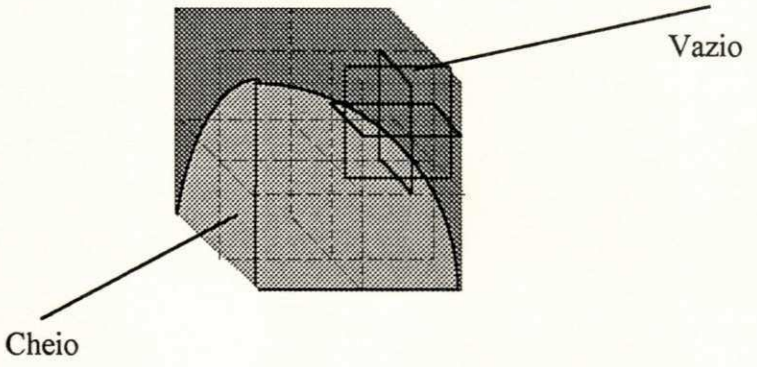


Figura 4. 16 - Terceiro passo da modelagem por octrees

Este processo de subdivisões termina quando todas as células são homogêneas para uma dada resolução.

Do mesmo modo que a quadtree, o modelo é armazenado em uma estrutura de árvore, onde cada nó representa um cubo disjunto dos outros e que expressa uma característica daquela região do espaço.

4.2.4.3.3) Octree linear

4.2.4.3.3.1) Introdução

A modelagem octree produz um modelo mais compacto do que a técnica de spatial occupancy enumeration (Ex: cuberrile) para representar um objeto. Esta compactação decorre do fato de que na octree podemos ter cubos (octantes) de tamanhos maiores do que a "resolução" que estamos utilizando, o que equivale a juntarmos voxels (octante de tamanho mínimo) vizinhos em um cubo maior, que aglutine estes voxels, sem perda de resolução.

Embora a octree convencional já proporcione um modelo mais compacto, este modelo, em alguns casos, ainda é muito grande devido a complexidade do objeto e o grau de precisão necessário para o modelo (Figura 4. 17). Para resolver este problema foi criada a octree linear.

Volume	Tamanho	Num. níveis da octree	Numero de nós interme-diários	Número de nós folhas	Total de nós
coração	128X128X128	7	47457	332200	379657
cabeça	128X128X128	7	174405	1220836	1395241
joelho	128X128X128	7	246615	1726306	1972921

Figura 4. 17 - Tabela com tamanhos de octrees [Aman85]

4.2.4.3.3.2) O que é

A octree linear é uma forma de codificação da octree convencional que reduz a necessidade de 9 campos, normalmente

requeridos para armazenar um nó de uma octree convencional, para um único código (Figura 4. 18).

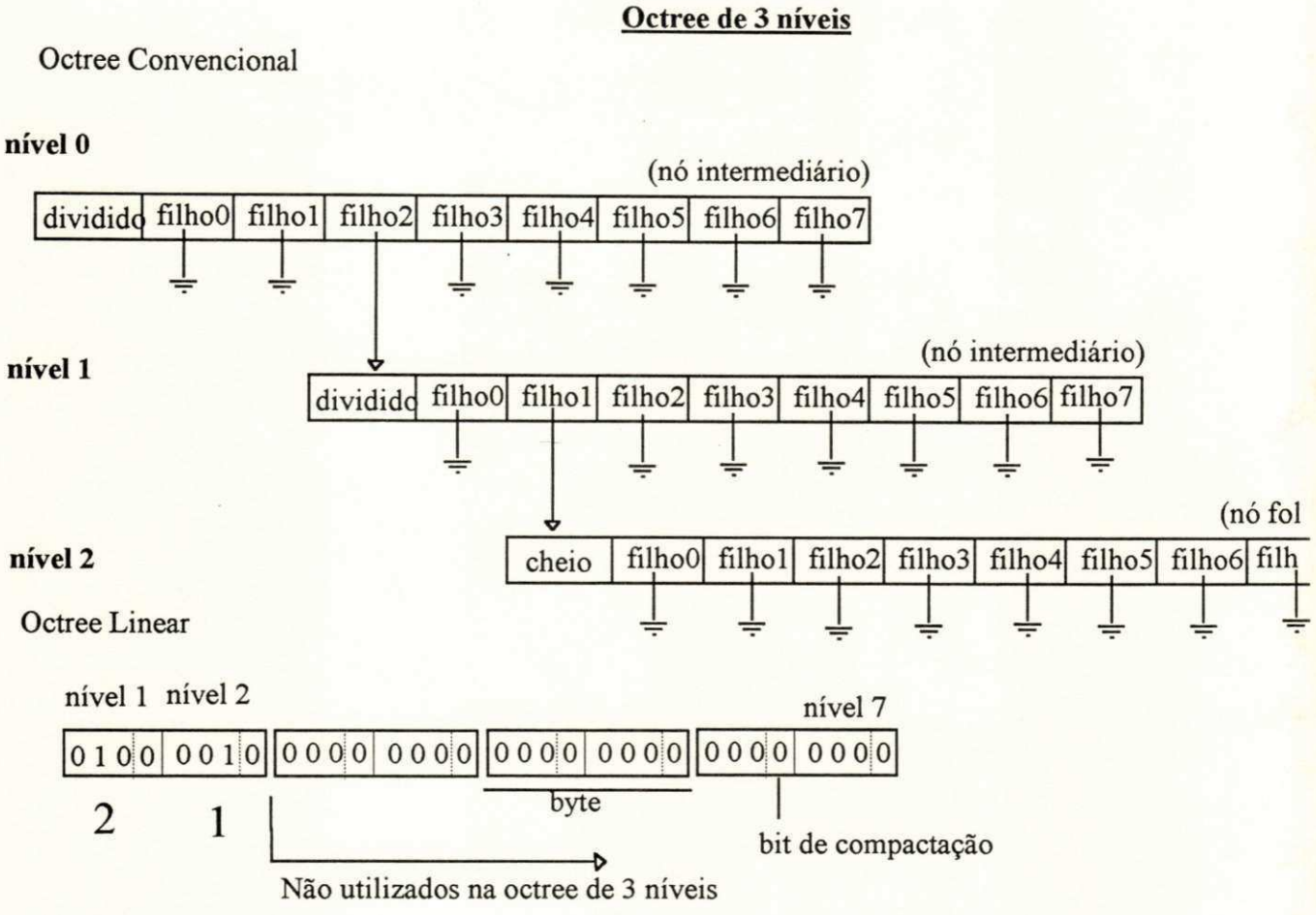


Figura 4. 18 - Octree linear & octree convencional

Em termos de bytes, utilizando 1 byte para armazenar os atributos (cheio, vazio, dividido) e 4 bytes para guardar o apontador de um nó do próximo nível da árvore, os 9 campos (1 de atributos, 8 ponteiros), ocupariam 33 bytes para representar um determinado nó; ao passo que o código que representa um nó folha, na octree linear (da forma como implementada neste trabalho) ocupa 4 bytes em uma octree com até 8 níveis de subdivisões.

Outra informação importante é que na octree linear, nós só armazenamos informações relativas aos nós folhas da octree que

estão cheios (Figura 4. 19), não armazenando nenhuma informação dos nós vazios (que ficam subentendidos pela ausência de um nó) ou nós intermediários, que indicam o relacionamento entre os nós folhas (estas informações ficam diluídas no código que representa o nó cheio). Este fato nos dá um grau de economia de memória ainda maior, quando comparado a octree convencional.

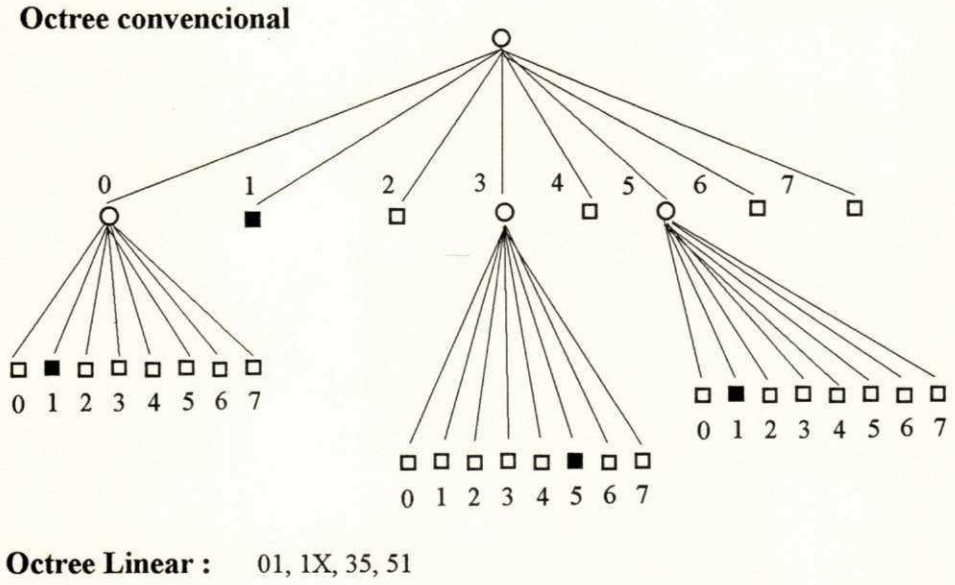


Figura 4. 19 - Representação linear de uma octree convencional

4.2.4.3.3.3) Como codificar

Como introdução, é bom salientar que a codificação, realizada por simples inspeção da octree não linear, é uma coisa inviável em termos de tempo (para construir duas octrees) e memória (para armazená-las). Logo não é admissível a construção de uma octree para depois encontrar a sua forma linear; por isto um outro método deve ser adotado.

Vamos tentar descrever como foi realizada a codificação da octree neste trabalho.

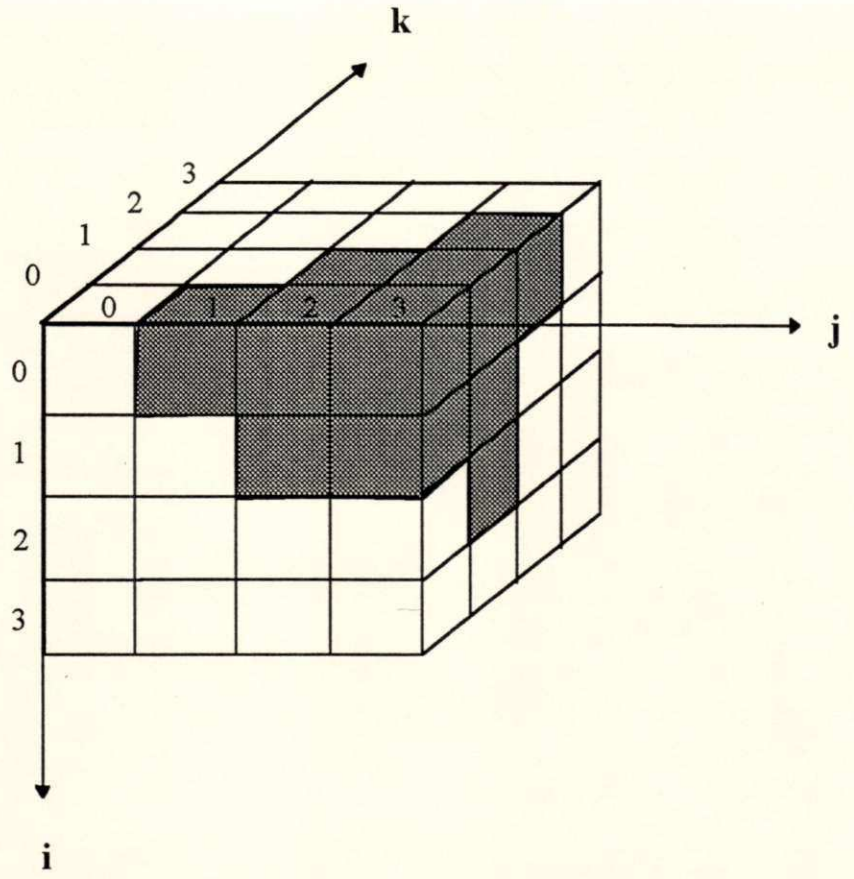


Figura 4. 24 - Descrição de um voxel

b.2) Descrição de um octante da octree linear

Um octante, em uma octree linear, é descrito por uma seqüência hierárquica de dígitos octais (código), onde os dígitos mais significativos representam os octantes maiores onde o octante está inserido (Figura 4. 25). O último dígito, o menos significativo, representa o octante propriamente dito (Figura 4. 25).

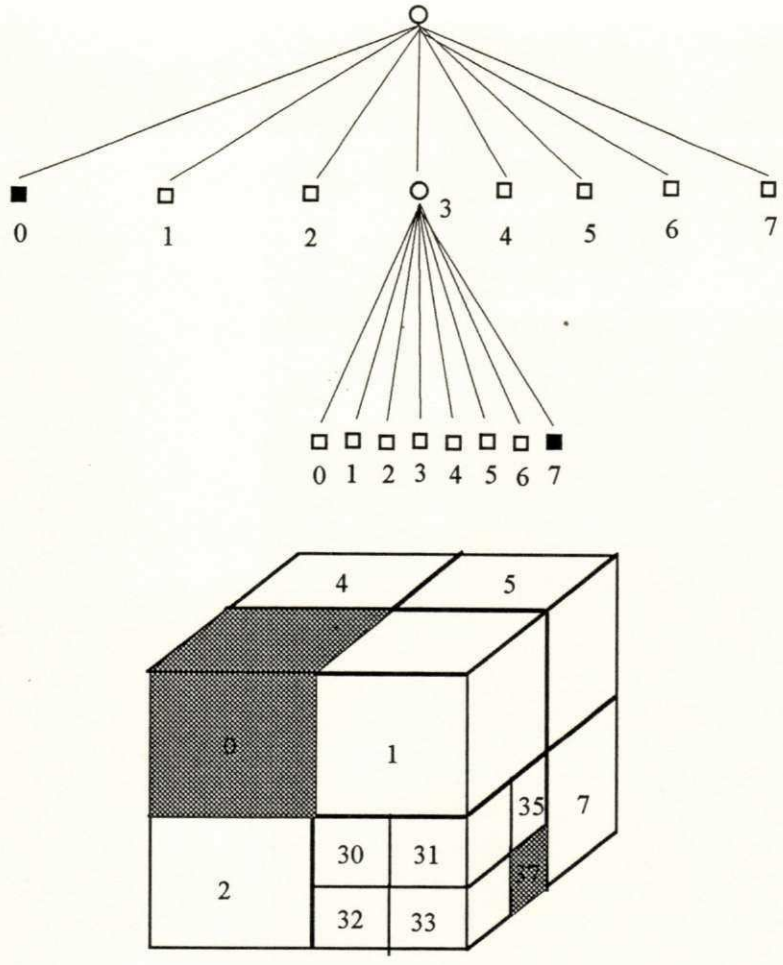


Figura 4. 25 - Descrição de um octante

Do parágrafo anterior conclui-se que, para podermos codificar um octante da octree, precisamos estabelecer uma numeração para cada um dos 8 octantes de um determinado nível da octree[5].

Convencionando-se uma numeração, podemos representar um elemento da octree por um número octal

$$Q = q_{n-1} \cdot 8^{n-1} + q_{n-2} \cdot 8^{n-2} + \ldots + q_1 \cdot 8^1 + q_0 \cdot 8^0$$

[5]Em nosso caso utilizamos a mesma numeração do artigo [Garg81].

onde cada dígito "q" identifica um octante, em uma ordem hierárquica ao qual o elemento pertence (Figura 4. 25).

b.3) Conversão de uma descrição para a outra

Para mapear um voxel representado por suas coordenadas, (i, j, k) (i, j, k = 0, 1, ..., 2^n) que descrevem a posição do voxel no espaço, para um octante, podemos seguir os seguintes passos :

b.3.1) Encontrar a forma binária de (i, j, k)

$$i = d_{n-1} \cdot 2^{n-1} + \ldots + d_1 \cdot 2^1 + \ldots + d_0 \cdot 2^0$$
$$j = c_{n-1} \cdot 2^{n-1} + \ldots + c_1 \cdot 2^1 + \ldots + c_0 \cdot 2^0$$
$$k = e_{n-1} \cdot 2^{n-1} + \ldots + e_1 \cdot 2^1 + \ldots + e_0 \cdot 2^0$$

b.3.2) Para determinar os "q's" que compõem o código do octante, devemos olhar as seguintes condições :

Se $e_{n-1} = d_{n-1} = c_{n-1} = 0$, nós sabemos que o voxel está no octante ¨0¨ para o nível ¨n-1¨ da octree, logo $q_{n-1} = 0$; assim sucessivamente :

Se $e_{n-1} = 0, d_{n-1} = 0$ e $c_{n-1} = 1$, então $q_{n-1} = 1$;

Se $e_{n-1} = 0, d_{n-1} = 1$ e $c_{n-1} = 0$, então $q_{n-1} = 2$;

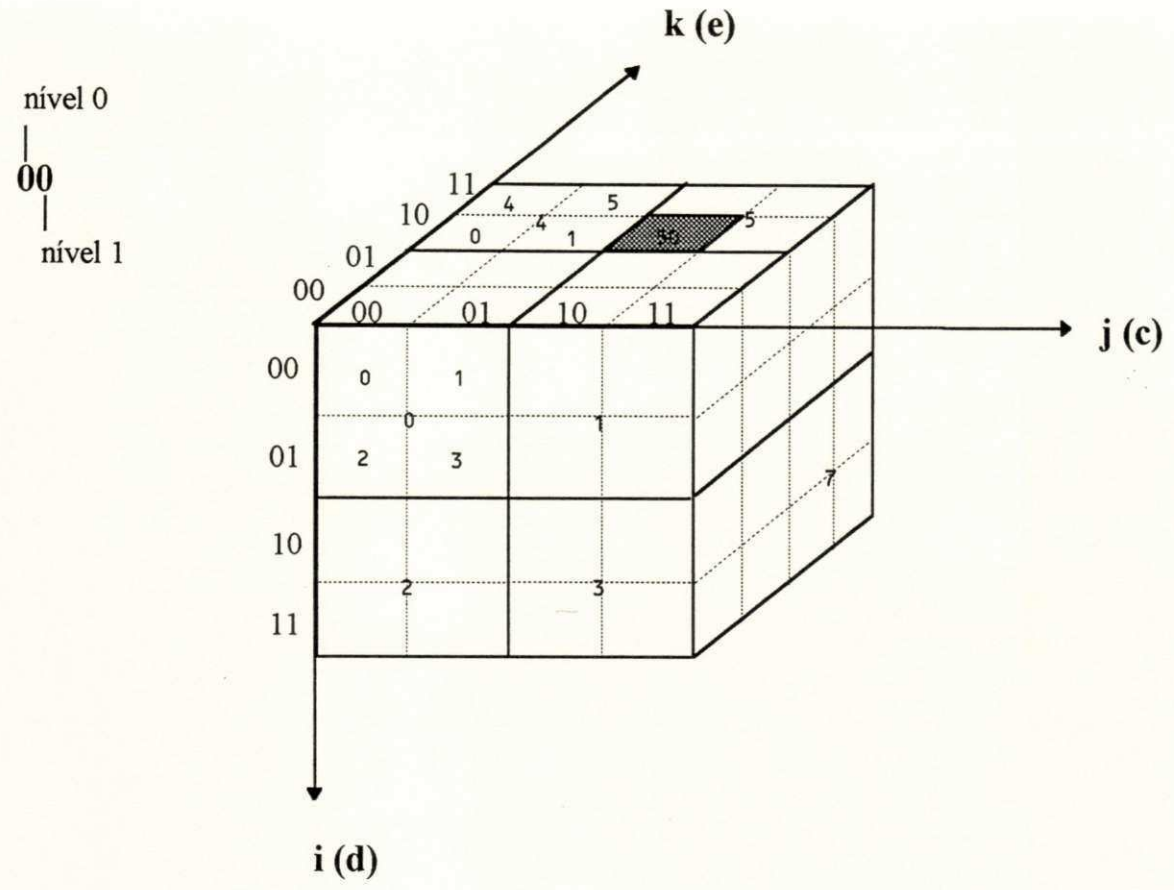
Se $e_{n-1} = 0, d_{n-1} = 1$ e $c_{n-1} = 1$, então $q_{n-1} = 3$;

Se $e_{n-1} = 1, d_{n-1} = 0$ e $c_{n-1} = 0$, então $q_{n-1} = 4$;

Se $e_{n-1} = 1, d_{n-1} = 0$ e $c_{n-1} = 1$, então $q_{n-1} = 5$;

Se $e_{n-1} = 1, d_{n-1} = 1$ e $c_{n-1} = 0$, então $q_{n-1} = 6$;

Se $e_{n-1} = 1, d_{n-1} = 1$ e $c_{n-1} = 1$, então $q_{n-1} = 7$;



	k (e)	i (d)	j (c)	
decinal	2	0	2	
binário	1	0	1	5
	0	0	0	0
				código do octante

Figura 4. 26 - Conversão de uma descrição para a outra

Assim, de acordo com a numeração que escolhemos, podemos notar que :

$$q_1 = e_1 . 2^2 + d_1 . 2^1 + c_1 . 2^0$$

Estendendo a soma anterior para todos os dígitos de (i, j, k) obtemos o código completo do elemento :

j	$c_{\frac{n-1}{2}} \cdot 2$	...	$c_i \cdot 2^2$	...	$c_0 \cdot 2^2$
	+		+		+
i	$d_{\frac{n-1}{2}} \cdot 2$	...	$d_i \cdot 2^2$	...	$d_0 \cdot 2^2$
	+		+		+
k	$e_{n-1} \cdot 2^2$	...	$e_i \cdot 2^2$	...	$e_0 \cdot 2^2$
	⇓		⇓		⇓
(código)	$\mathbf{q_{n-1}}$	...	$\mathbf{q_i}$	...	**q0**

A decodificação é feita pelo processo inverso, onde cada dígito é decomposto para uma dada precisão n, remontando os dígitos binários de (i, j, k).

c) Passo 2 ("compactação" da octree)

Como dissemos antes, a octree obtida neste processo não está em sua forma mais compacta, pois todos os octantes que a compõem são de tamanho mínimo, o que não aconteceria em uma octree convencional, onde encontramos octantes de tamanhos maiores (Figura 4. 19).

A partir desta octree linear "descompactada", podemos obter uma outra octree mais compacta, Irene em seu artigo [Garg81] sugere que façamos o seguinte :

c.1) 8 octantes de nível "n", ao longo de um octante de nível "n+1", podem ser agrupados pela troca do dígito relativo ao nível do voxel por uma marca que denote o agrupamento (Figura 4. 27).

Octantes								Agrup.	
100	101	102	103	104	105	106	107	→	10**X**
110	111	112	113	114	115	116	117	→	11**X**
120	121	122	123	124	125	126	127	→	12**X**
130	131	132	133	134	135	136	137	→	13**X**
140	141	142	143	144	145	146	147	→	14**X**
150	151	152	153	154	155	156	157	→	15**X**
160	161	162	163	164	165	166	167	→	16**X**
170	171	172	173	174	175	176	177	→	17**X**
									↓
									1**XX**

Figura 4. 27 - Agrupamento de octantes

Este processo de agrupamento pode ser repetido sempre que encontremos 8 octantes filhos de um mesmo pai.

Os códigos da octree linear são mantidos ordenados para facilitar a pesquisa na octree e tornar o processo de compactação mais simples (isso conclui o processo de modelagem).

CAPÍTULO 5

PROJEÇÃO

5.1) Introdução

Os objetos do mundo real sempre possuem 3 dimensões. O nosso modelo, que se propõe a representar objetos reais, armazena informações destas 3 dimensões; mas a tela de vídeo, que é o lugar onde o modelo será exibido, possui somente 2 dimensões.

Para resolvermos o problema, modelo 3D e vídeo 2D, foram criadas técnicas de projeção que transformam objetos 3D em projeções 2D.

Este decréscimo de dimensão deve seguir alguns objetivos[Leve68]:

a) Preservar ao máximo, na projeção, as características do objeto 3D (Proporção entre arestas e ângulos);

b) Dar uma visão espacial do modelo, facilitando a tarefa de evocar mentalmente o objeto real;

c) Criar um efeito visual próximo ao de uma fotografia, introduzindo um grau de realismo na representação 2D.

5.2) Tipos de projeção

Faremos agora uma breve exposição sobre alguns tipos de projeções (Figura 5. 1), nos atendo ao caso planar, com a finalidade de promover um certo embasamento e explicar os motivos que nos levaram a escolher a projeção ortogonal (ou ortográfica) para realizar este trabalho.

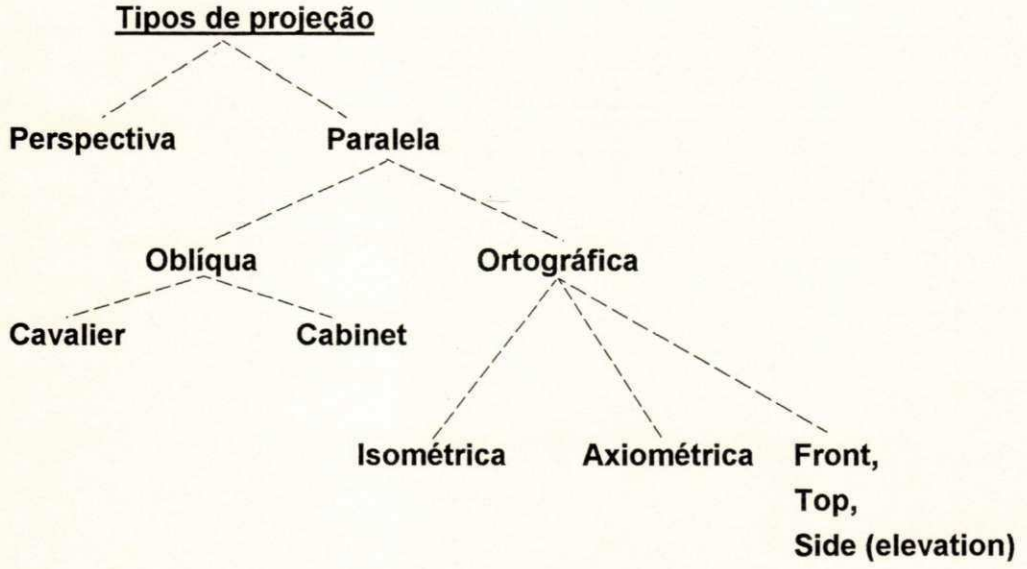


Figura 5. 1 - Tipos de projeção

Projeções planares podem ser divididas em duas classes: projeções perspectivas e projeções paralelas. O que diferencia um tipo de projeção do outro é a posição do centro de projeção em relação ao plano de projeção. Na Figura 5. 2 "S" é o centro de projeção para onde convergem os projetores, que são linhas que ligam pontos no objeto ao centro de projeção. A projeção de cada ponto se dá na intercessão do projetor com o plano de projeção.

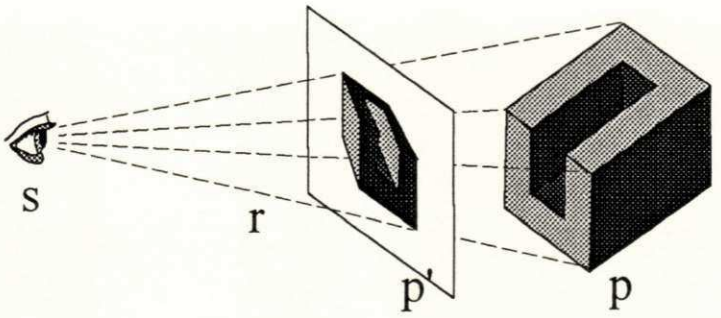


Figura 5. 2 - Projeção em perspectiva

Caso o centro de projeção esteja a uma distância finita do plano de projeção, esta projeção é chamada de perspectiva, caso esta distância seja infinita (especificado apenas a direção de projeção) esta será chamada de paralela (Figura 5. 3).

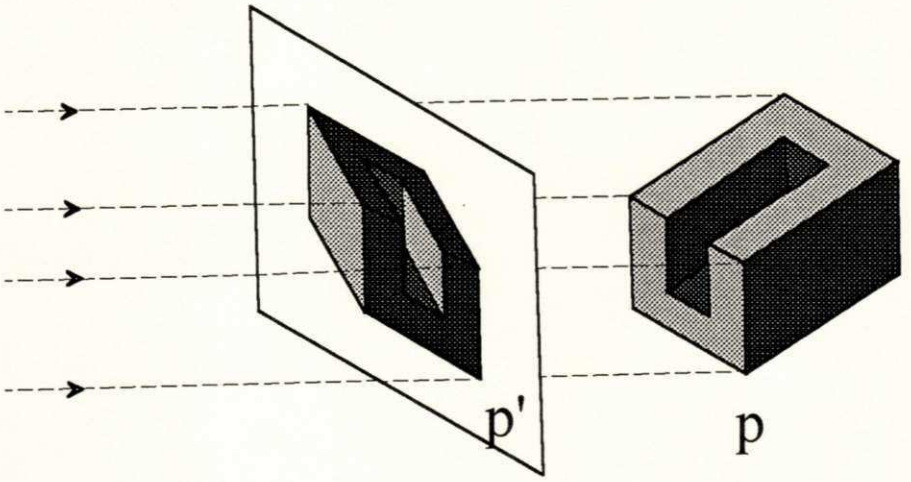


Figura 5. 3 - Projeção paralela

5.2.1) Projeções em perspectiva

Existem muitos tipos de projeções em perspectiva. O fator básico de distinção entre elas se dá com relação ao número de pontos de fuga principais[1].

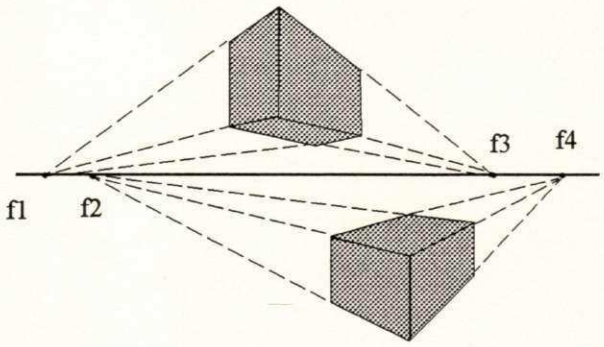


Figura 5. 4 - Pontos de fuga

Caso este conjunto de linhas sejam paralelas a um dos eixos principais (Xr, Yr, Zr), este ponto de fuga é chamado de ponto de fuga principal.

As projeções em perspectiva são classificadas de acordo com o número de pontos de fuga principais que elas possuem.

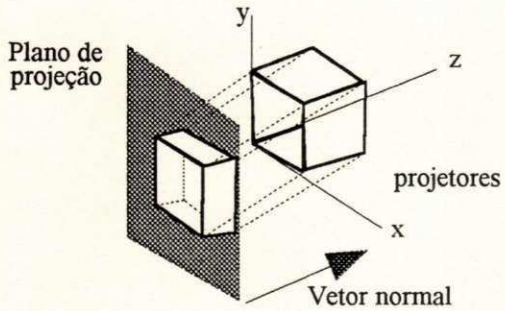
O resultado da projeção perspectiva, devido às deformidades que produz, torna a visualização do objeto mais simples [Leve68]. Por outro lado, estas deformidades provocam mudanças nos ângulos e tamanhos das arestas, o que pode ser um problema caso desejemos realizar alguma atividade que venha a necessitar destas medidas. Assim, a projeção perspectiva tem um valor mais estético do que prático, por isso não optamos por este tipo de projeção.

---

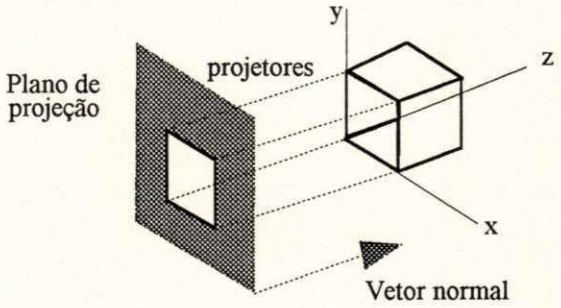
[1]Um conjunto de linhas paralelas que convergem para um ponto em uma projeção perspectiva, determinam um ponto chamado de ponto de fuga (Figura 1. 4).

5.2.2) Projeções paralelas

As projeções paralelas podem ser divididas em duas classes: ortográficas e oblíquas, dependendo da relação entre a direção de projeção e o plano de projeção. Caso a direção de projeção seja perpendicular ao plano de projeção, temos uma projeção ortográfica ou ortogonal, caso contrário, ela será oblíqua (Figura 5. 5).



Projeção Oblíqua



Projeção Ortográfica

Figura 5. 5 - Projeções : oblíqua e ortográfica

5.2.2.1) Projeções oblíquas

Existem muitos tipos de projeções oblíquas. Como exemplo deste grupo, citaremos dois tipos de projeção:

Cavalier - neste tipo, a direção de projeção e o plano de projeção formam um ângulo de 45 graus, o que faz com que o tamanho original das linhas, que são perpendiculares ao plano de projeção, seja mantido (Figura 5. 6).

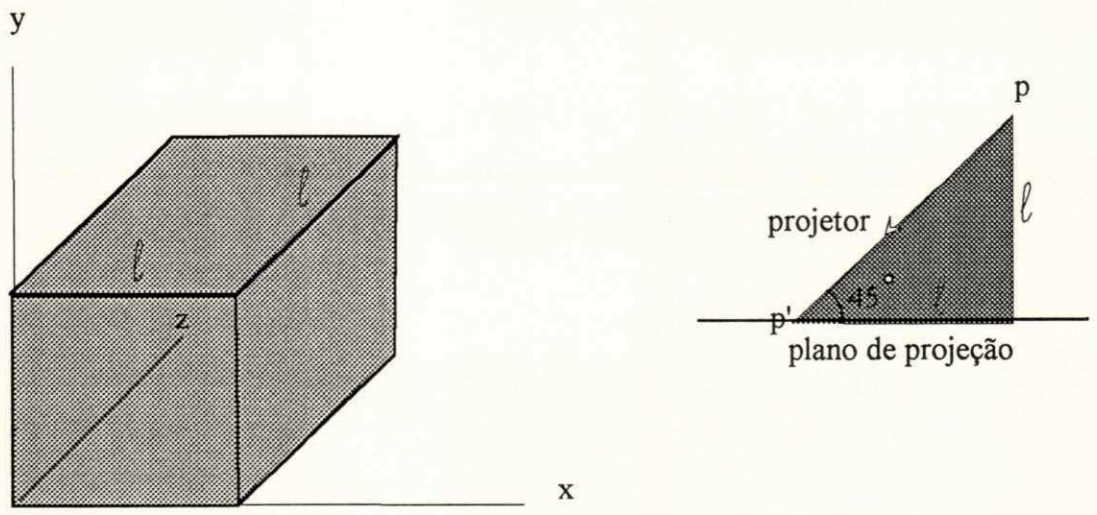


Figura 5. 6 - Projeção cavalier

Cabinet - neste outro tipo, a direção de projeção e o plano de projeção, fazem um ângulo igual a Arctan(2) = 63,4, assim as linhas perpendiculares ao plano de projeção são reduzidas à metade (Figura 5. 7).

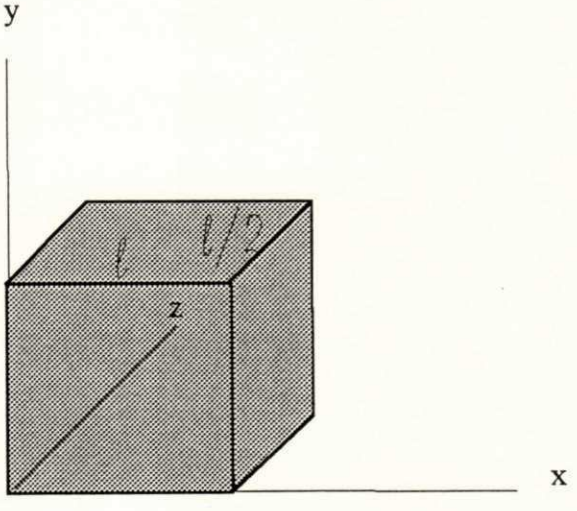


Figura 5. 7 - Projeção cabinet

Como não percebemos nenhuma vantagem significativa que nos forçasse a escolher a projeção oblíqua, foi adotada a projeção ortográfica pelos motivos abaixo:

a) Na projeção ortográfica, definida a direção de projeção, automaticamente definimos o plano de projeção, pois este é perpendicular à direção de projeção, o que não acontece na projeção oblíqua, implicando na necessidade do usuário informar mais um parâmetro ou mais algum cálculo para o próprio software.

b) Facilidade em definir escalas para realizar medidas, de acordo com a posição do plano de projeção [Fole82].

c) Grande utilização na área técnica, principalmente em engenharia [Leve68].

5.2.2.2) Projeções ortográficas

Este é o tipo de projeção onde o plano de projeção é perpendicular à direção de projeção (Figura 5. 5). As projeções ortográficas são classificadas de acordo com a posição do plano de projeção. Neste trabalho a forma como o usuário especifica a posição do plano de projeção permite a ele obter qualquer tipo de projeção ortográfica, como será mostrado no decorrer do capítulo.

Quando o plano de projeção é também perpendicular a um dos eixos principais temos as projeções do tipo: FRONT-ELEVATION, TOP-ELEVATION, SIDE-ELEVATION (Figura 5. 8). Estes tipos de projeções permitem boa medição de ângulos e comprimentos, sendo muito utilizadas em engenharia; no entanto, existe a dificuldade de visualizar todo o objeto, dado que estamos sempre vendo o objeto de um conjunto limitado de posições.

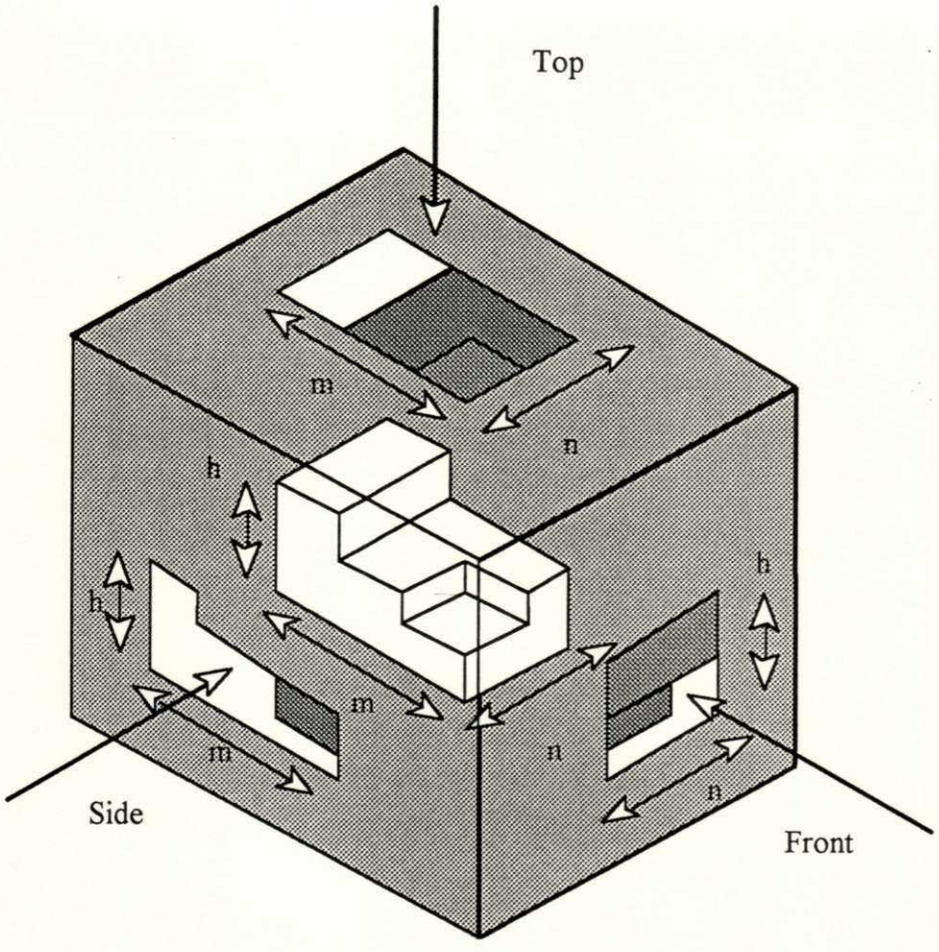


Figura 5. 8 - Projeções : top, side e front elevation

Axiométricas - são projeções onde o plano de projeção não é perpendicular a nenhum eixo principal (Figura 5. 9). Isto permite visualizar muitas faces do objeto ao mesmo tempo, o que nos dá uma idéia melhor da tridimensionalidade do objeto. Nesta projeção o paralelismo das linhas é mantido, as arestas podem ser medidas com fatores de escala e alguns ângulos são preservados;

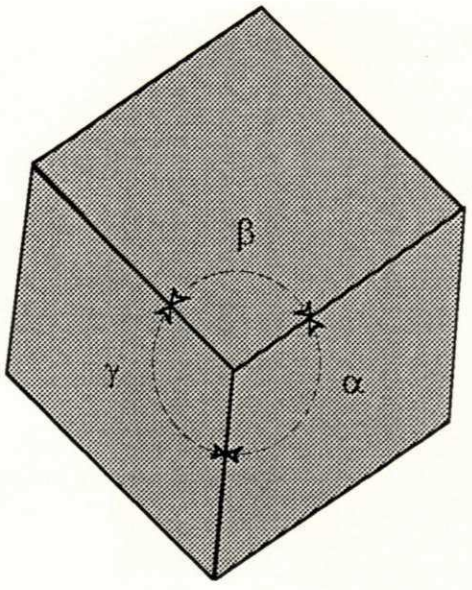


Figura 5. 9 - Projeção axiométrica

Isométricas - nestas o plano de projeção forma ângulos iguais com os 3 eixos principais (Figura 5. 10). A vantagem em relação ao anterior é a de podermos utilizar um só fator de escala para realizar as medidas.

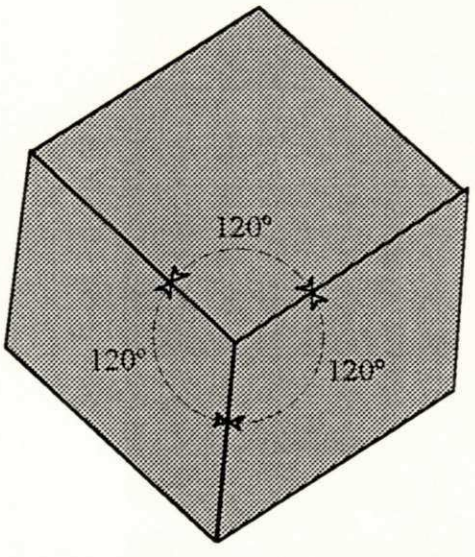


Figura 5. 10 - Projeção isométrica

5.3) Especificação de uma vista

Após a escolha do tipo de projeção, para que possamos realizá-la, três elementos devem ser especificados segundo [Fole82]:

a) volume de visualização - delimita qual a porção dentro do universo que será projetada;

b) plano de projeção ou janela de projeção - especifica onde a porção do universo, delimitada pelo volume de visualização, será projetada;

c) janela de visualização - faz um mapeamento de coordenadas do plano de projeção para coordenadas da tela (dispositivo de saída).

Vamos agora explicar como estes elementos foram implementados neste trabalho.

Primeiro é necessário que sejam definidos dois conceitos, que serão utilizados no restante do capítulo:

a) Octante do modelo - é um octante codificado e armazenado na lista linear e representa uma parte do objeto;

b) Octante do universo - esta é uma forma de referenciar os planos que delimitam um octante específico, não obrigatoriamente pertencente ao modelo do objeto.

5.3.1) Volume de visualização

O volume de visualização é delimitado pelo octante do universo de nível 0, o que implica que temos sempre todo o objeto dentro do volume visual. Caso se deseje observar parte do objeto, pode-se

lançar mão do plano de recorte, que é um recurso a disposição do usuário. Na prática o volume de visualização inexiste, o que é justificável dado que só temos um objeto a visualizar de cada vez.

5.3.2) Plano de projeção

Neste trabalho utilizamos 3 sistemas de coordenadas (Figura 5. 11) :

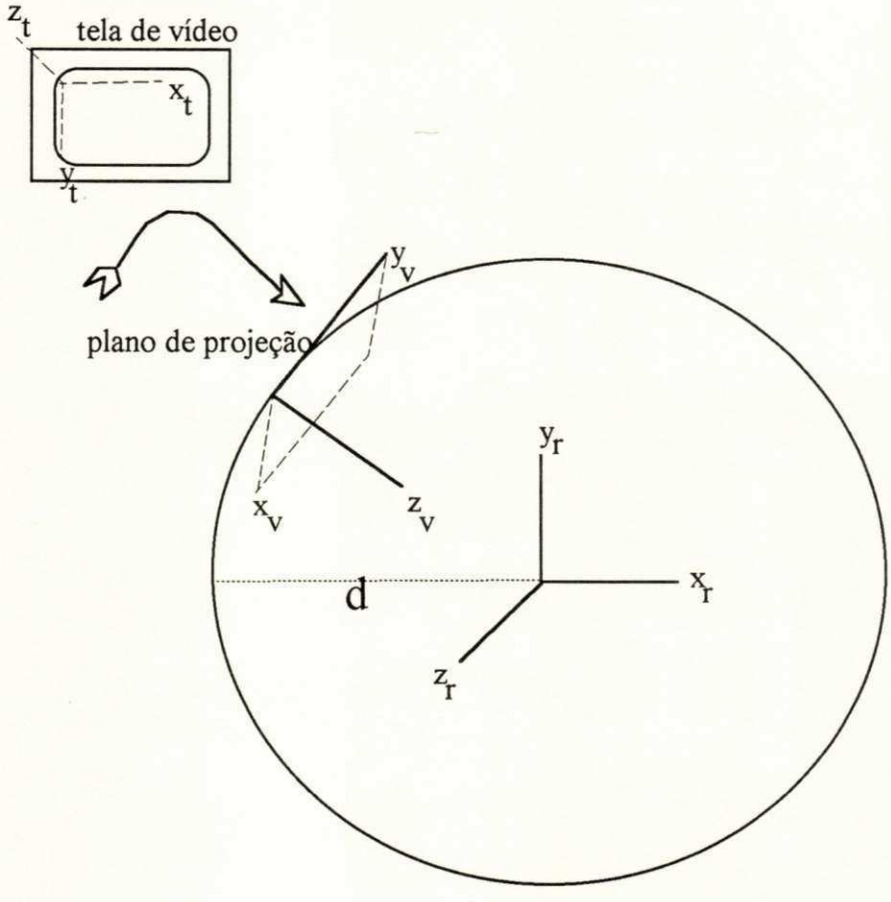


Figura 5. 11 - Sistemas de coordenadas

a) sistema real ou universal (x_r, y_r, z_r) - é o sistema para o qual o nosso modelo foi gerado. Este sistema não muda de posição (é fixo) e serve como base para que posicionemos todos os outros elementos.

b) sistema visual (x_V, y_V, z_V) - é o sistema utilizado para projetar o objeto. Este sistema tem sempre seu eixo z_V apontando para a origem do sistema de coordenadas reais, pois o objetivo aqui é sempre projetar o objeto que se encontra na origem (Figura 5. 11);

O plano onde projetamos o objeto é formado pelos outros dois eixos: x_V e y_V;

A origem deste sistema está sempre a uma distância fixa "d" da origem do sistema de coordenadas reais. Esta distância fixa foi adotada para que, na geração do z buffer (como será explicado), não tivéssemos distâncias de sinais contrários[2] , e também tivéssemos distâncias com tamanhos mínimos, o que não ocorre quando o plano está a uma distância maior do que esta.

Sua posição[3], em relação ao sistema de coordenadas do universo, está definida através da informação de azimute e elevação.

O azimute é medido em graus, e especifica uma rotação do sistema visual em torno do eixo y_r do sistema real (Figura 5. 12).

---

[2]Quando o plano de projeção intercepta o objeto temos distâncias de sinais contrários.

[3]Neste trabalho os eixos x_r x_V e y_r y_V são sempre co-planares, porque não há rotação em torno de z_r.

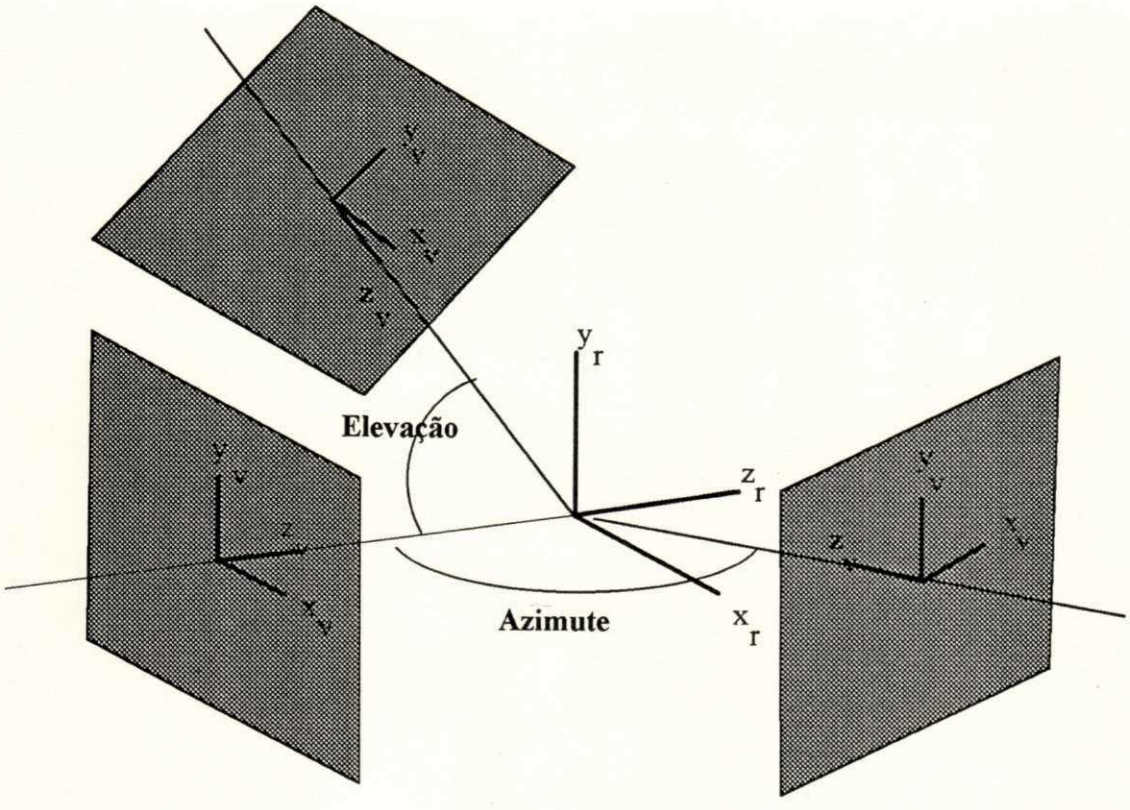


Figura 5. 12 - Sistema de coordenadas visual

A elevação também é medida em graus. Ela especifica uma rotação de todo o sistema de coordenadas visuais em torno do eixo x_r do sistema de coordenadas reais (Figura 5. 12).

c) sistema de tela (x_t, y_t, z_t) (faz o papel da janela de visualização) - este sistema encontra-se posicionado na tela da seguinte forma: sua origem está situada no canto superior esquerdo da tela, seu eixo z_t cresce para dentro da tela, o eixo x_t tem o sentido crescente para a direita, e y_t para baixo (Figura 5. 11).

5.4) Remoção de superfícies ocultas (z buffer)

Quando se está trabalhando na geração de imagens 2D com um certo grau de realismo é necessário, para simular a opacidade de um objeto, evitarmos que sejam exibidas partes do objeto que não seriam normalmente visíveis a partir da vista especificada.

Este processo de remoção pode ser extremamente oneroso em tempo, devido ao grande número de comparações que necessitam ser realizadas para que possamos definir as porções visíveis do objeto [Fole82]. Este problema se torna mais grave quando estamos utilizando uma modelagem por octree linear, onde o objeto é modelado por um número muito grande de pequenas primitivas[4] (octantes), com forma de acesso basicamente seqüêncial (pois não guardamos informações para um acesso mais elaborado).

Assim sendo, devemos ao máximo evitar algoritmos que tenham que percorrer a estrutura (modelo) muitas vezes, seja parcial ou totalmente.

Na busca desta redução foram adotadas técnicas de projeção e remoção de superfícies ocultas, que pudessem ser executadas simultaneamente, ou seja, ao realizarmos a projeção devemos realizar a remoção destas superfícies, ao invés de realizar isto em passos separados.

Existem basicamente dois grupos de algoritmos para remoção de superfícies ocultas chamados "object space" e "image space".

5.4.1) Object space

O primeiro grupo chamado "object space" procura analisar o conjunto de objetos que compõem uma cena e determinar as porções visíveis de cada objeto, para depois exibir estas partes.

---

[4]Para que se tenha uma idéia da dimensão disto, basta que tomemos como exemplo uma modelagem do crânio que, segundo [Aman85], possui 1.220.836 octantes.

Como nesta abordagem o processo de remoção se dá em duas etapas, ela não será utilizada neste trabalho, embora a mesma apresente uma vantagem em relação à adotada, que é a de manter independência entre o objeto e a resolução em que este será apresentado, o que permite ampliações sem perda de qualidade.

5.4.2) Image space

Esta segunda abordagem foi a utilizada neste trabalho. Ela procura determinar quais as partes visíveis da cena para cada pixel na tela ou em um buffer de projeção.

Esta estratégia pode ser divida em dois outros grupos, de acordo com a forma utilizada para determinar o atributo do pixel :

a) Partindo de cada pixel, determinar o seu atributo, o que nos levaria intuitivamente ao algoritmo de "ray tracing";

b) Simplesmente projetar os objetos, e a partir da projeção de cada objeto, ir determinando os novos atributos dos pixels. O que nos levaria intuitivamente a pensar em algoritmos de z buffer, como classicamente conhecidos[5] .

5.4.2.1) Z buffer

Também conhecido como "depth buffer", foi desenvolvido por catmull [catm74] para ser implementado tanto em software quanto em hardware, devido a sua simplicidade.

---

[5]O termo Z buffer, às vezes, é utilizado na literatura para falar do algoritmo de [Catm74], outras vezes é utilizado só para descrever uma estrutura de dados, na qual armazenamos as distâncias de pontos do plano de projeção até o objeto.

A técnica deste algoritmo[6] consiste em testar a visibilidade das superfícies ponto a ponto, através da comparação das distâncias dos pontos das superfícies em relação ao plano de projeção. Este algoritmo, como projetado por [catm74], necessita de dois buffers. O primeiro para armazenar atributos de cor dos objetos da cena, chamado C buffer. O segundo buffer para armazenar a distância de cada ponto do plano de projeção até o objeto, chamado z buffer (estrutura de dados). Em nossa implementação, dado que temos um único objeto, o buffer para atributos de cor se torna desnecessário.

No primeiro passo do algoritmo, o z buffer (estrutura de dados) é inicializado para um valor máximo de distância. Em seguida, cada polígono (em uma ordem arbitrária) é submetido a uma conversão de rastreio ("scan conversion"), que transforma a superfície do objeto em um conjunto de pixels. Durante o processo de conversão, para cada ponto calculado, olhamos se este possui uma distância do plano de projeção Z1, menor do que a distância Z2 já armazenada no z buffer para aquele ponto. Se isto for verdade, então armazenamos uma nova distância para Z2.

Neste trabalho o processo de conversão de rastreio ("scan convesion") foi sensivelmente simplificado através da adoção de duas medidas:

a) associação da projeção do menor octante, com as dimensões de um pixel na tela;

b) todos os octantes que não possuem o tamanho mínimo são subdivididos recursivamente em seus 8 filhos, até obtermos

---

[6]O termo Z buffer, às vezes, é utilizado na literatura para falar do algoritmo de [Catm74], outras vezes é utilizado só para descrever uma estrutura de dados, na qual armazenamos as distâncias de pontos do plano de projeção até o objeto.

c) facilidade na produção de imagens de diferentes tamanhos e resoluções;

d) como característica negativa temos o consumo de muito tempo de processamento, o que cria uma barreira para que muitas pessoas o utilizem[8].

No processo de RAY TRACING existem basicamente duas fases: a primeira é a determinação e envio de um raio, e a segunda trata de determinar a intercessão deste raio com o objeto.

[Whit80] diz que a segunda fase é a mais dispendiosa e pode chegar a consumir até 95% do tempo de processamento do algoritmo[9].

Faremos agora uma descrição geral do algoritmo projetado, bem como uma exposição das otimizações adotadas em cada fase, omitindo algumas partes e detalhes de implementação que não são relevantes para o entendimento do algoritmo.

---

[8]O esforço desta parte do trabalho foi o de encontrar técnicas é estratégias que pudessem melhorar o desempenho destes algoritmos, bem como o de mostrar a sua viabilidade para uso em máquinas de pequeno porte (neste caso um PC-80386).

[9]Neste trabalho tentamos otimizar todas as partes que compõem este algoritmo, devido às limitações da plataforma de hardware disponível PC-80386.

```
Início
    Inicializa o Z buffer para o valor infinito;
    Solicita parâmetros do usuário;
    Determina a projeção do universo;
    Para todo ponto na projeção do universo faça :
    Início
        Determinar a intercessão do raio que sai deste ponto
        com o objeto;
        Se existe intercessão então
            Guardar a distância da intercessão no z buffer;
    Fim
Fim
```

A fase "determina a projeção do universo" procura definir um polígono de projeção do octante do universo de nível zero, que funciona como uma "bound box" para o nosso objeto (Figura 5. 13).

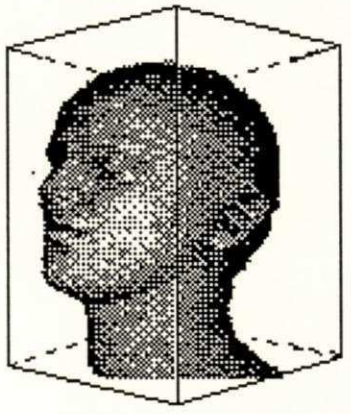


Figura 5. 13 - Octante do universo de nível zero

Esta fase foi criada para reduzir o número de raios que precisariam ser calculados e traçados, pois sabemos que todo objeto está incluso no octante do universo de nível zero, logo a projeção do objeto também está inclusa na projeção deste octante, desta forma, basta que analisemos somente os raios que partem do interior desta projeção.

A determinação desta projeção é feita da seguinte forma:

- Sabemos que existem basicamente 3 tipos de polígonos de projeção de um octante, de acordo com a quantidade de componentes que possui o vetor normal ao plano de projeção (Figura 5. 14).

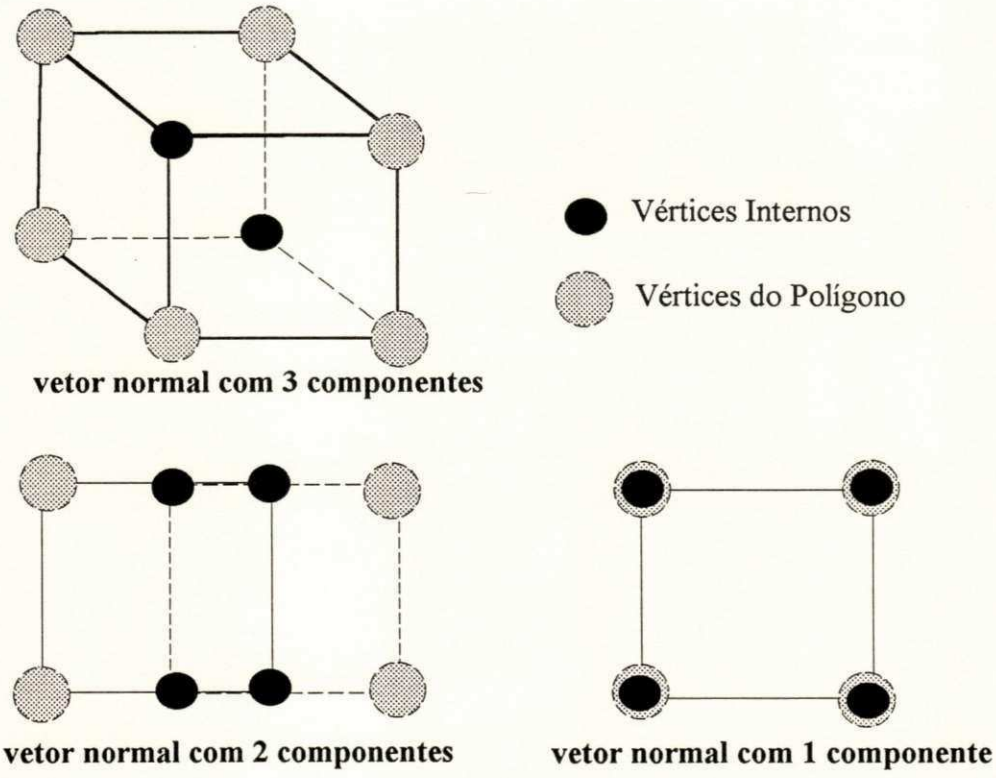


Figura 5. 14 - Polígonos de projeção

- Desta forma podemos detectar quais os vértices que fazem parte do polígono de projeção e quais serão internos a este (Figura 5. 14). Assim projetamos os vértices do polígono, e determinamos as arestas que compõem o polígono de projeção.

Após a determinação deste polígono, podemos excluir uma boa parte dos raios, que seguramente não tocarão o objeto.

Vamos descrever agora a parte de "determinação da intercessão do raio com o objeto". Esta fase do algoritmo é um pouco mais complexa e foi dividida em duas etapas distintas: cálculo matemático da intercessão e pesquisa no modelo, que detalharemos agora.

5.4.2.2.1) Cálculo matemático da intercessão

Como sabemos, um objeto modelado através de uma octree é composto de uma grande quantidade de cubos (octantes), assim, nosso problema de intercessão se resume em determinar a intercessão de uma reta (raio[10]) com um cubo.

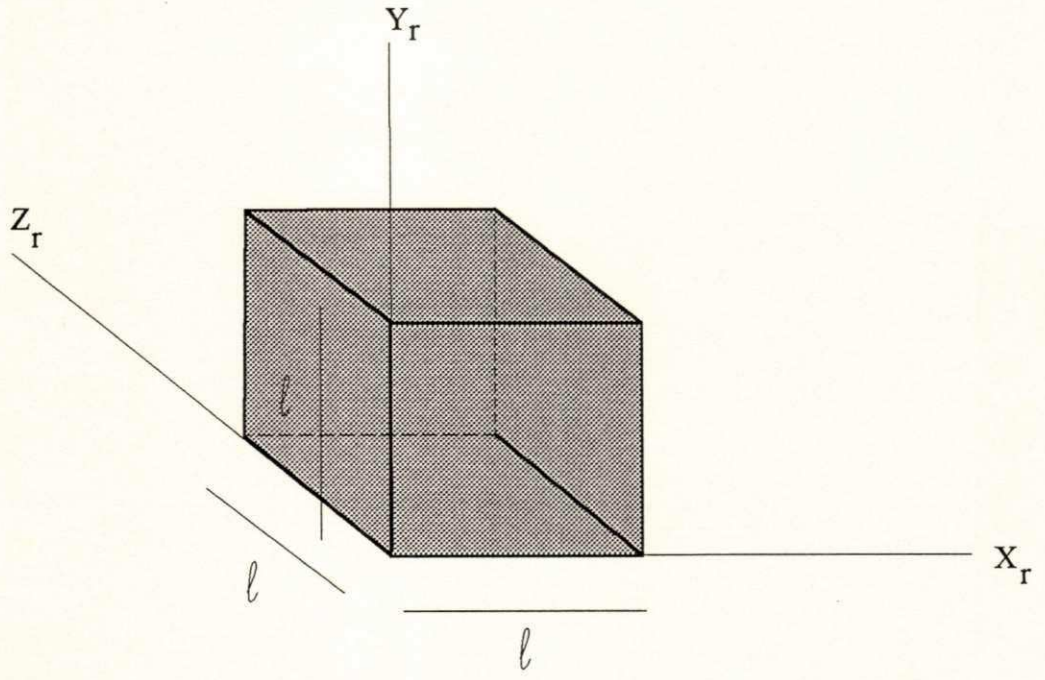


Figura 5. 15 - Orientação das faces de um octante

[10]Um raio fica bem determinado através de um ponto, dentro do polígono de projeção e o vetor normal ao plano de projeção, bastando assim percorrer os pontos que estão dentro do polígono para determinar os raios.

Um cubo é composto de 6 faces planas, cada uma destas é paralela a um plano do tipo A=<constante>, onde "A" é um dos eixos principais (Xr, Yr, Zr) do sistema de coordenadas do universo (Figura 5. 15). Este fato torna mais simples a determinação da intercessão de um raio com um destes planos, pois basta que analisemos a componente do raio na direção do plano desejado. Vejamos a equação paramétrica de um raio, dado um ponto p (x0, y0, z0) no plano de projeção e uma normal N (a, b, c):

$$xr = x0 + at$$
$$yr = y0 + bt$$
$$zr = z0 + ct \text{ com: } t \in R$$

suponhamos um plano x' = k, que é um plano do mesmo tipo dos planos que delimitam um octante.

para encontrar a intercessão do raio com este plano, basta que tomemos

$$xr = x0 + at \text{ para } xr = k \text{ assim } t = (k - x0) / a .$$

Foi visto até agora como calcular a intercessão de um dos planos do cubo com um raio. Agora veremos como determinar a intercessão do raio com o cubo propriamente dito.

Para descobrir esta intercessão, primeiro devemos analisar a intercessão do raio com os planos que compõem o cubo, segundo cada uma das 3 direções do R3 (Figura 5. 16).

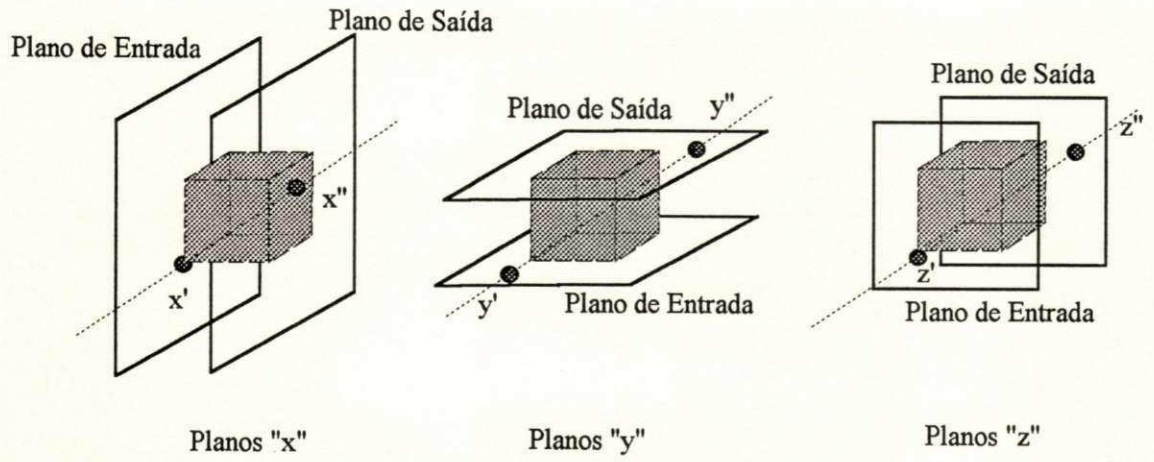


Figura 5. 16 - Intercessão de um raio com os planos delimitadores de um octante

Podemos assim perceber que existe uma faixa de valores para "t" onde o raio está passando entre os dois planos daquela direção(Figura 5. 17).

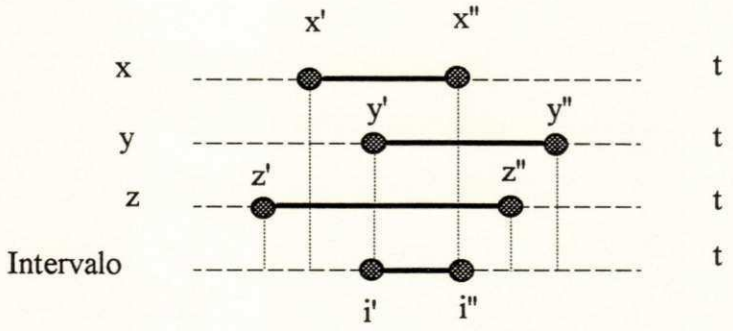


Figura 5. 17 - Intervalo de intercessão

A intercessão dos valores de todas as faixas nos dá o intervalo de intercessão [i', i"] onde o raio está passando entre os 6 planos, logo dentro do cubo (Figura 5. 17).

Assim, sempre que este intervalo estiver definido, existe a intercessão do raio com o cubo e vice-versa.

Podemos também separar as 6 faces do cubo em dois grupos: faces de entrada, que são as faces frontais ao plano de projeção, e

faces de saída, que são as faces que ficam na parte posterior do cubo em relação ao plano de projeção[11] (Figura 5. 16).

Analisando agora mais detalhadamente cada intervalo, podemos perceber que o limite inferior do intervalo é dado pela intercessão do raio com o plano de entrada naquela direção e o limite superior pela intercessão do raio com o plano de saída (Figura 5. 17).

Para o intervalo de intercessão, podemos perceber que o seu limite inferior (ponto de entrada no cubo) é dado pelo maior dos limites inferiores dos intervalos. Assim, cada direção que possua um limite inferior igual ao limite inferior do intervalo de intercessão, terá o seu plano de entrada perfurado pelo raio no momento da entrada[12].

Logo :

a) se somente um intervalo possui seu limite inferior igual ao do intervalo de intercessão é porque o raio perfura uma face do cubo;

b) se dois intervalos possuem seus limites inferiores iguais ao do intervalo de intercessão é porque o raio perfura duas faces do cubo, ou seja uma aresta;

c) se três intervalos possuem seus limites inferiores iguais ao do intervalo de intercessão é porque o raio perfura um vértice do cubo.

---

[11]Como os octantes estão dispostos todos com uma mesma orientação, as faces de entrada e saída para um deles são as mesmas para todos.

[12]A mesma análise é válida para o ponto de saída do cubo.

5.4.2.2.2) Algoritmo de busca no modelo

Como estamos utilizando uma octree linear, temos pouca informação para facilitar a pesquisa de um elemento neste modelo.

A primeira iniciativa, para melhorar esta busca, foi a de ordenar os elementos do modelo possibilitando que se realizasse um pesquisa binária na octree[13].

Desta forma, foi atingido um desempenho na pesquisa equivalente ao de termos nossa octree linear armazenada em uma árvore binária [Knut73].

Mesmo assim, com a utilização, esta técnica se mostrou ineficiente; a solução foi montar um tabela "hashing"[14] para agilizar ainda mais esta pesquisa.

Nossa tabela "hashing" consiste basicamente em um array de elementos do tipo (limite inicial, limite final), onde cada elemento delimita uma área da octree, na qual podemos encontrar os octantes que possuem códigos com os 3 primeiros dígitos "iguais" ao do índice da tabela (Figura 5. 18).

---

[13]A pesquisa binária tem como pior caso (maior número de acesso à estrutura) $\log_2 n$ acessos, onde n é o número de elementos na octree [knut73].

[14]Para maiores esclarecimentos sobre tabela "hashing" procurar [knut73].

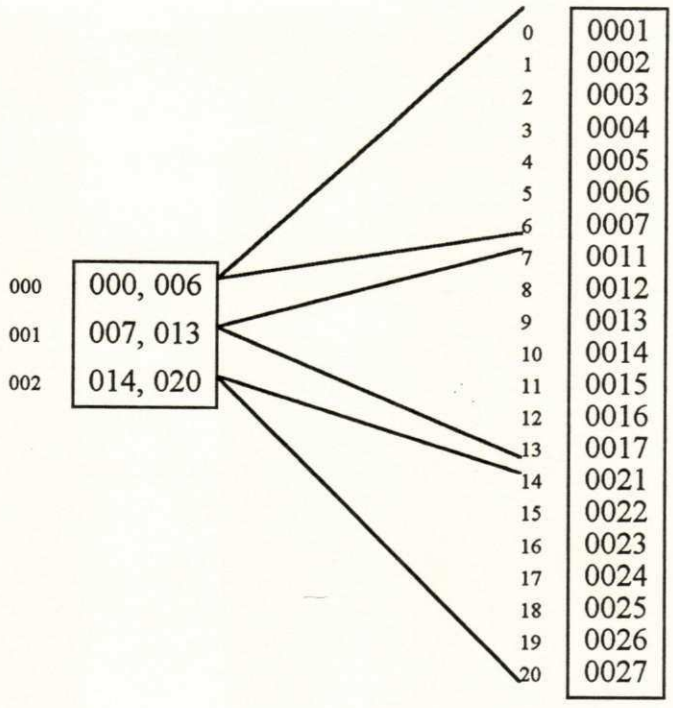


Figura 5. 18 - Tabela hashing

Para acessar a octree, através desta estrutura, devemos fazer o seguinte:

a) Separar os 3 primeiros dígitos do código que queremos encontrar e montar um índice (pela concatenação dos dígitos) para uma das 512 (8!3) entradas de nossa tabela (Figura 5. 19).

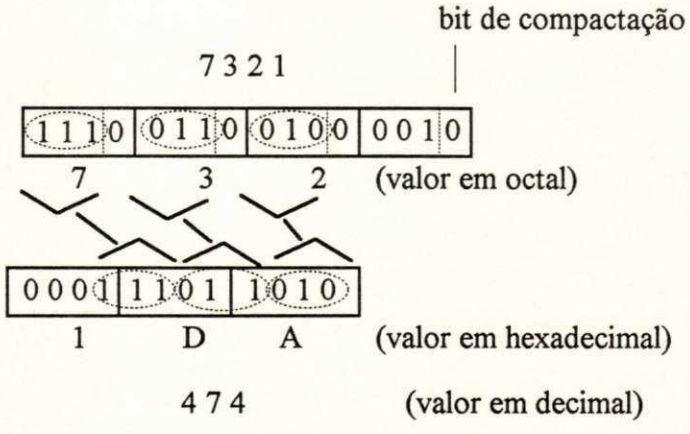


Figura 5. 19 - Montagem do índice

b) Com este índice encontramos um elemento na tabela que informa dentro de que limite na octree está o elemento procurado.

c) Após esta determinação dos limites, fazemos uma pesquisa binária nesta área para encontrar o elemento.

Com esta tabela, além de reduzirmos o número de acessos necessários para encontrar um elemento (o que equivale a trabalhar com uma octree virtualmente pequena), ainda temos a vantagem de: caso um octante de nível 3, ou menor, não exista no modelo, a entrada na tabela estará vazia, evitando assim o acesso a octree[15].

Agora, de posse destas informações (estratégia de busca, cálculo matemático da intercessão), vamos tentar descrever dois algoritmos que fazem uso destas para determinar a intercessão de um raio com o objeto.

5.4.2.2.3) Perfuração direta

A idéia básica deste algoritmo é encontrar o octante do universo de tamanho mínimo (maior nível), que é tocado pelo raio, quando este perfura o octante do universo de nível zero.

A partir deste octante de tamanho mínimo, ir descobrindo os próximos octantes mínimos do universo, que são também perfurados, até encontrar um octante que exista no modelo ou o raio sair do octante do universo de nível zero (Figura 5. 20).

---

[15]O que facilitará o algoritmo de perfuração direta, que tem como pior caso a inexistência da intercessão com o objeto, como será visto à frente.

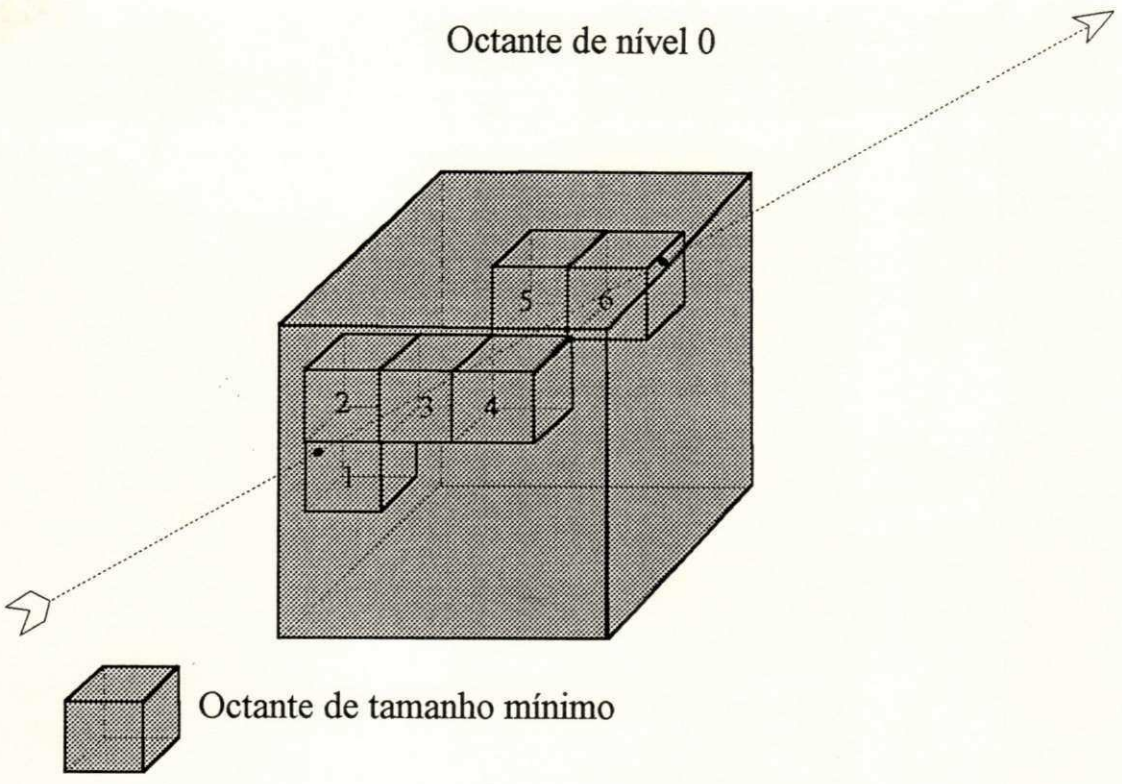


Figura 5. 20 - Travessia do raio

Como podemos notar, este algoritmo tem o seu pior caso quando surge um raio que passa por todo octante do universo de nível zero sem tocar o objeto.

```
Início
    Encontrar o ponto de entrada do raio no octante do universo de
    nível zero;
    Codificar o octante do universo de tamanho mínimo que é
    perfurado neste ponto;
    Enquanto raio dentro do "nível zero" e octante perfurado não
    encontrado no modelo faça :
    Início
        Encontrar ponto de saída deste octante;
        Codificar o próximo octante do universo de tamanho
        mínimo que foi perfurado;
    Fim
Fim
```

O ponto de entrada no octante do universo de nível zero é obtido a partir do cálculo de intercessão dos planos de entrada deste octante com o raio. Vale ressaltar que, como o raio é traçado de dentro da projeção dele próprio, a intercessão é garantida.

Encontrado o ponto de entrada, dividimos as suas coordenadas pelo tamanho do lado do octante de tamanho mínimo, assim conseguimos saber qual o octante do universo foi perfurado e realizar a sua codificação.

O ponto de saída do raio, através de um octante, pode ser uma de duas coisas, o ponto de entrada no próximo octante ou o ponto de saída do octante do universo de nível zero. Uma otimização no cálculo do ponto de saída pode ser realizada a partir do segundo octante determinado; isto se deve ao fato de que alguns planos que delimitam um octante, também delimitam o próximo octante perfurado, isto pode ser descoberto analisando se o ponto de saída pertence a uma face, aresta ou vértice do octante (Figura 5. 20). Assim não é necessário recalcular a intercessão do raio para estes planos que se mantém.

5.4.2.2.4) Teste hierárquico de intercessão

A idéia básica deste outro algoritmo é a de testar hierarquicamente a intercessão. Estes testes são realizados com os octantes do universo, que funcionam com "bound boxes". Caso exista a intercessão com um octante do universo, testamos se este octante existe no modelo. Se existir, calculamos a distância do ponto de intercessão. Se existir um descendente do octante procurado, subdividimos este octante em seus 8 filhos e repetimos o processo para cada um destes filhos.

Algoritmo :

```
Início
    Empilha (código do octante do universo de nível zero);
    Enquanto a pilha não está vazia faça :
    Início
        Desempilha( código);
        Testa intercessão( código, raio );
        Se intercessão existe então
            Se existe descendentes então
                Empilha os oito filhos do nó
            Se não, Se existe o octante no modelo então
            Início
                Guarda distância de intercessão;
                Esvazia pilha;
            Fim
    Fim
Fim
```

CAPÍTULO 6

ILUMINAÇÃO

6.1) Introdução

Iluminar é calcular a intensidade de luz, que deve ser observada para cada ponto da imagem, de acordo com a sua posição, características da superfície e algumas leis da óptica que se deseja simular [Hear86].

A iluminação procura atingir dois objetivos [Gord85] que se apresentam conflitantes[1] :

a) mostrar o objeto com o máximo de detalhes de sua superfície;

b) eliminar ou diminuir os defeitos na superfície, causados por deficiências na modelagem ou projeção.

A iluminação é geralmente colocada em uma fase separada no processo de síntese da imagem; principalmente porque ela pode exigir ajustes e recálculos que se tornariam inviáveis caso tivéssemos que refazer todo trabalho já realizado até aquele ponto.

Para realizar a iluminação, precisamos definir duas coisas :

---

[1]Estes objetivos são conflitantes porque normalmente quando enfatizamos um deles o outro também é enfatizado e vice-versa.

a) modelo de iluminação - tem por objetivo simular algumas leis da óptica;

b) método de iluminação - preocupa-se em estabelecer uma estratégia para aplicar o modelo de iluminação definido à toda a imagem;

Estes passos serão abordados em detalhes nos itens 3 e 4 deste capítulo.

6.2) Características desejáveis

Existem 3 características que devemos procurar obter na iluminação:

a) ampla aplicabilidade - possibilidade da técnica aplicada, ser utilizada em um maior grupo de situações possíveis, o que permitirá certa independência desta fase de geração da imagem em relação às fases anteriores;

b) boa qualidade da imagem - a qualidade da imagem produzida está intimamente ligada a dois fatores: precisão da segmentação do objeto[2] e precisão dos vetores normais calculados para as superfícies[3];

c) baixo esforço computacional - dependendo do tempo consumido nas fases anteriores e do número de vezes que temos de

---

[2]Esta não é matéria de estudo deste capítulo e segundo [hoeh86] não é matéria de estudo em computação gráfica.

[3]Segundo [hoeh87] a precisão das normais é a principal responsável pela qualidade da imagem.

repetir esta fase, esta característica se torna mais ou menos importante.

6.3) Modelo para determinação de intensidade luminosa

6.3.1) Tipos de fontes de luz (dimensão da fonte)

6.3.1.1) Fontes pontuais

Uma fonte é dita pontual quando suas dimensões são desprezíveis em relação ao objeto iluminado. Assim, podemos considerar que toda a luz proveniente desta fonte sai de um único ponto (Figura 6. 1).

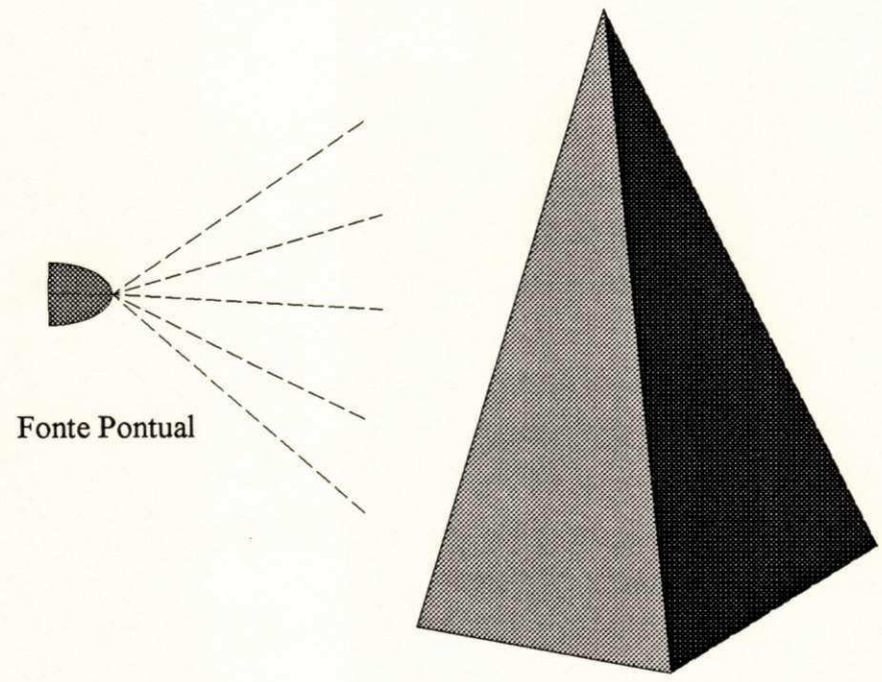


Figura 6. 1 - Fonte pontual

6.3.1.2) Fontes extensas

Uma fonte é dita extensa quando o seu tamanho não pode ser desconsiderado em relação ao tamanho do objeto por ela iluminado (Figura 6. 2).

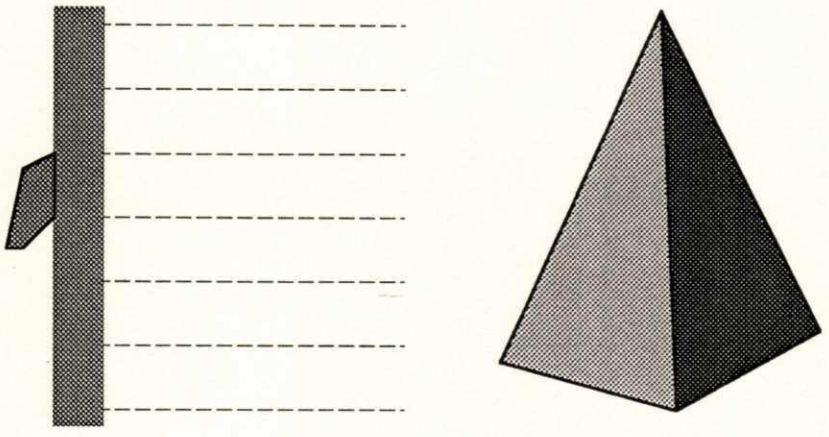


Figura 6. 2 - Fonte extensa

6.3.2) Tipos de fontes de luz (natureza da luz)

6.3.2.1) Fontes de luz primárias

São as fontes que emitem luz própria, como por exemplo o sol.

6.3.2.2) Fontes de luz secundárias

São os objetos do meio que iluminados refletem a luz, podendo assim funcionar como uma fonte de luz para outros objetos.

A combinação das múltiplas reflexões da luz, na vizinhança do objeto iluminado, produz uma iluminação uniforme chamada de iluminação ambiental ou de fundo.

6.3.3) Tipos de reflexão

6.3.3.1) Reflexão difusa

Dizemos que a reflexão é difusa quando a luz refletida é "espalhada" em todas as direções (Figura 6. 3), devido às características da superfície (rugosidade, tipo de material, etc...).

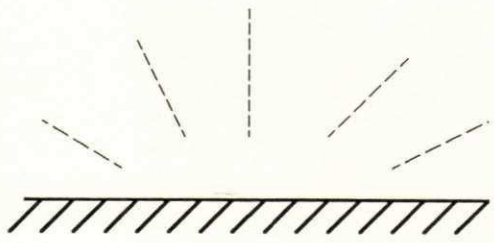


Figura 6. 3 - Reflexão difusa

Um modelo matemático para calcular a intensidade da reflexão difusa[4] em um ponto da superfície pode ser encontrado em [Hear86].

$$I_d = K_d \cdot I_a + ((K_d \cdot Ip) / (d + d_0)) \cdot (\vec{N} \cdot \vec{L})$$

com:

K_d - Coeficiente de reflexão difusa (característico da superfície);

I_a - Intensidade de luz ambiental;

Ip - Intensidade da luz proveniente de uma fonte de luz;

d - Distância do objeto à fonte de luz; (ponto)

d_0 - Constante acrescida para evitar que o denominador da divisão chegue a zero, para pontos muito próximos da fonte de luz;

---

[4]O trabalho descrito no artigo de [gord85] utiliza somente reflexão difusa, neste trabalho deixamos disponível modelos para reflexão difusa e especular.

$\vec{N}$ - Vetor normal à superfície (no ponto);
$\vec{L}$ - Vetor direção da fonte de luz;
($\vec{N}$. $\vec{L}$) - Cosseno do ângulo entre os vetores;

Explicação do modelo :

K_d . I_a - Calcula a componente da intensidade de luz difusa, devido a iluminação ambiental (que é constante em todas as direções);

((K_d . Ip) / (d + d_0)) . ($\vec{N}$. $\vec{L}$) - Calcula a componente da reflexão difusa, devido a uma fonte de luz primária[5] [6].

Segundo uma das leis da óptica, a intensidade luminosa decresce com o quadrado da distância entre a fonte de luz e a superfície iluminada. Na prática o fator (1 / d^2) não produz um decréscimo natural de luz, isto porque as fontes de luz natural apresentam propriedades intermediárias entre fontes pontuais e fontes extensas de raios paralelos [Fole82]. Assim, é utilizado o fator (1 / d) que produz um decaimento mais lento [Gord85].

6.3.3.2) Reflexão especular

A reflexão especular ocorre quando uma superfície reflete toda luz incidente, produzindo um ponto brilhante na imagem [Hear86]. Para superfícies opticamente perfeitas, o ângulo de incidência e o ângulo de reflexão são iguais (Figura 6. 4).

---

[5]Esta intensidade depende do ângulo entre o raio de luz incidente e a normal à superfície , segundo a lei dos cossenos de Lambert, o que é representado aqui por ($\vec{N}$. $\vec{L}$).

[6]Caso desejemos acrescentar mais fontes de luz, basta que somemos mais termos iguais a este.

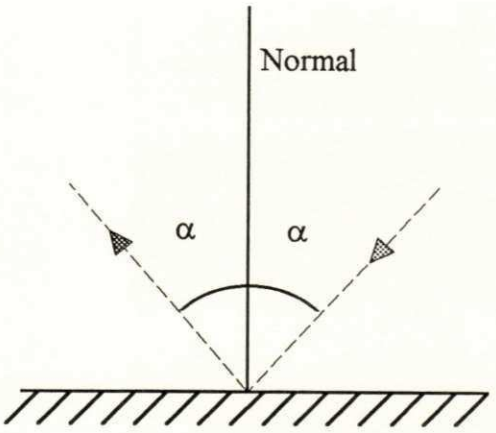


Figura 6. 4 - Reflexão especular

No caso de superfícies opticamente perfeitas, este efeito só pode ser visto na imagem, quando a direção de observação é a mesma do raio refletido (Figura 6. 5). Para objetos reais, a reflexão especular pode ser observada para uma determinada faixa de direções (Figura 6. 5).

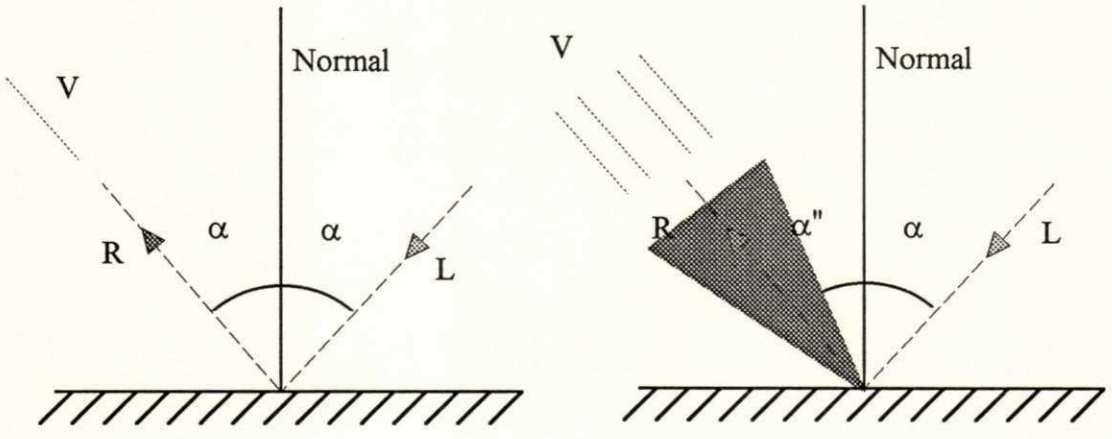


Figura 6. 5 - Reflexão especular em objetos reais

Phong [Buit75] desenvolveu um modelo para simular a reflexão especular. Em seu modelo, a reflexão especular é proporcional ao $\cos(\alpha)^n$, onde α é o ângulo formado pela direção de observação e a direção da fonte de luz. O valor atribuído para "n" determina o tipo

de superfície. Por exemplo: um valor elevado para "n" produz uma reflexão especular para uma pequena faixa de direções (Figura 6. 6). Simulando assim uma superfície com características próximas da superfície ideal (n ≥ 200).

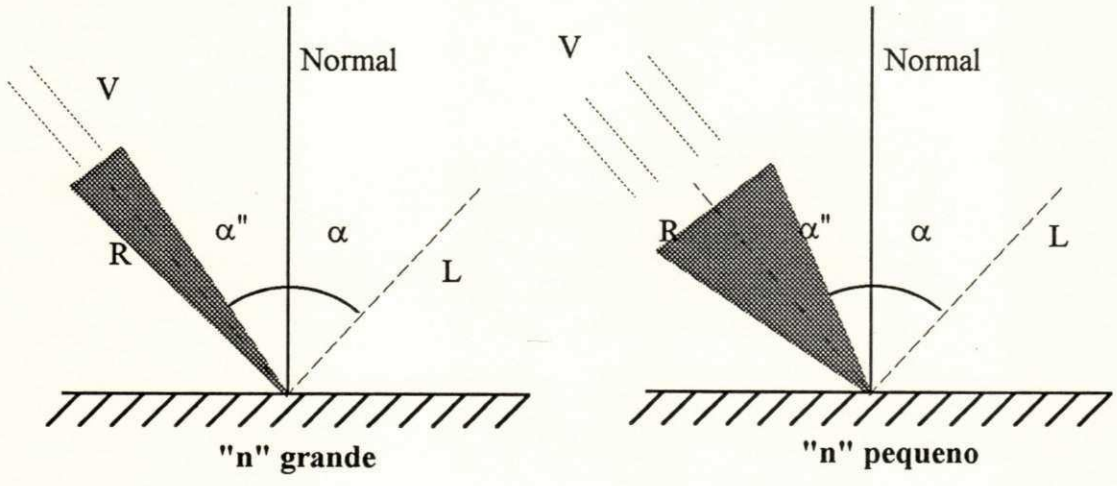


Figura 6. 6 - Comportamento do modelo de reflexão especular

A reflexão especular, em materiais reais, também depende do ângulo de incidência da luz [Fole82]. Em geral a intensidade de luz aumenta quando o ângulo aumenta. Para efeitos práticos, esta função (ângulo -> intensidade) pode ser simplificada para um valor constante K_S, que é determinado experimentalmente para o tipo de superfície.

Assim, acrescentando a reflexão especular ao modelo já descrito para a reflexão difusa, temos :

$$I = K_d \cdot I_a + (I_p / (d + d_0)) (K_d (\bar{N} \cdot \bar{L}) + K_S (\bar{V} \cdot \bar{R})^n)$$

6.4) Métodos de iluminação

Até agora vimos modelos matemáticos para calcular a intensidade de luz em um ponto. Este ponto pode pertencer a um objeto 3D, a polígonos projetados em uma superfície ou a uma pré-

imagem como o z buffer. Para podermos aplicar o modelo de iluminação a estas entidades e obtermos uma imagem iluminada, se faz necessária a adoção de um método de iluminação [Hear86].

Os métodos de iluminação podem ser divididos em dois grandes grupos: "object space" e "image space" [7].

A maioria dos métodos se diferenciam na maneira de calcular os vetores normais à superfície.

6.4.1) Métodos de iluminação para o espaço do objeto ("object space")

Este método, como o nome já indica, trabalha o modelo do objeto 3D e suas superfícies.

Quando as superfícies a serem iluminadas são aproximadas por polígonos, muitos métodos de iluminação estão disponíveis.

6.4.1.1) Distance only

Este método ignora totalmente a orientação da superfície no espaço e utiliza, para efeito de iluminação, somente a distância da superfície até a fonte de luz. Assim, neste método calcula-se a distância do ponto central da superfície até a fonte de luz, e a partir desta distância, calcula-se uma intensidade de luz que é atribuída a todos os pontos da face.

Este método produz imagens com uma aparência "plana", perdendo os pequenos detalhes de curvatura da superfície [Chen85]. O problema mais grave apresentado por este método está no fato de superfícies que tenham a mesma distância, independente de sua

---

[7]Ver capítulo de projeção.

orientação, aparecerem com uma mesma intensidade de luz; o que, algumas vezes, pode causar confusão na distinção destas superfícies.

6.4.1.2) Constant shading ou flat shading

Este método utiliza a normal da superfície para obter uma intensidade de luz que é estendida para todos os pontos do polígono. Esta normal pode já estar disponível no modelo ou ser calculada.

O principal problema deste método é encontrado, quando superfícies adjacentes possuem orientações muito distintas, o que produz um "malhado" desagradável na imagem [Hear86], além de continuarmos perdendo os pequenos detalhes de curvatura da superfície [Chen85].

6.4.1.3) Gouraud shading

Este método tenta criar a impressão de curvatura na superfície, por determinar a intensidade de luz para cada vértice do polígono, utilizando uma normal previamente calculada para cada vértice do polígono; em seguida, calcula a intensidade de luz para cada ponto da superfície a partir da interpolação das intensidades dos vértices.

Este método, por desprezar as variações no interior da superfície, pode atenuar ou mesmo omitir pontos de reflexão especular intensos, que ocorrem no interior das superfícies [Hear86].

Um outro problema está no fato de que a interpolação linear das intensidades pode produzir faixas brilhosas ou escuras na superfície chamadas de "mach bands" [Hear86].

6.4.1.4) Phong shading

O método de phong procura resolver as deficiências do método de "gouraud", calculando uma normal para cada ponto da superfície através da interpolação das normais dos vértices.

Segundo [Gord85], os problemas encontrados em "gouraud" continuam, só que menos pronunciados, além disto, o método de "phong" tem a inconveniência de ser muito demorado, como informado em [Chen85], [Gord85], [Hoeh87].

O artigo de [Gord85] ainda sugere duas desvantagens que são geralmente comuns a todos os métodos que trabalham no espaço do objeto:

a) Necessidade de armazenar informações adicionais como a vizinhança de um elemento ou as normais dos vértices de uma superfície (orientação da superfície), que para objetos muito complexos, tornariam o modelo muito grande e, conseqüentemente, ocuparia muita memória, o que nem sempre está disponível.

b) Objetos que venham a ser modificados pela aplicação de alguma operação, como por exemplo a segmentação (que será tratada no capítulo de recorte), terão que passar por um processo de recálculo destas informações adicionais, o que implicará em um consumo de tempo.

Estas novas informações, somadas às já mencionadas nos capítulos anteriores, vêm fortalecer a nossa decisão por trabalharmos no espaço da imagem.

6.4.2) Métodos de iluminação para o espaço da imagem ("image space")

Para que possamos utilizar os métodos de iluminação que trabalham no espaço da imagem, é necessário que já tenhamos produzido uma "pré-imagem", que em princípio pode ser uma imagem qualquer [Gord85], da qual possamos recuperar a informação da distância do ponto no objeto até o plano de projeção.

Segundo [Hoeh86] é bom lembrar que a qualidade da iluminação produzida está intimamente ligada à precisão que obtemos nas normais da superfície, que por sua vez serão calculadas a partir destas distâncias.

Assim, é recomendável que esta pré-imagem não tenha sido trabalhada por outros algoritmos de iluminação pois isto poderia gerar dificuldades na determinação das distâncias, o que comprometeria a qualidade de nossa imagem.

O ideal é que esta pré-imagem tenha, para cada pixel, apenas a distância do ponto visível no objeto até o plano de projeção (como é o caso do z buffer), ou ela seja uma imagem do tipo "distance only", onde esta informação é facilmente recuperável.

Segundo [Chen85] uma grande vantagem deste método é possuir complexidade proporcional ao número de pixels na imagem e não ao número de faces dos objetos da cena.

Vamos agora descrever dois métodos de iluminação que foram utilizados neste trabalho[8] :

---

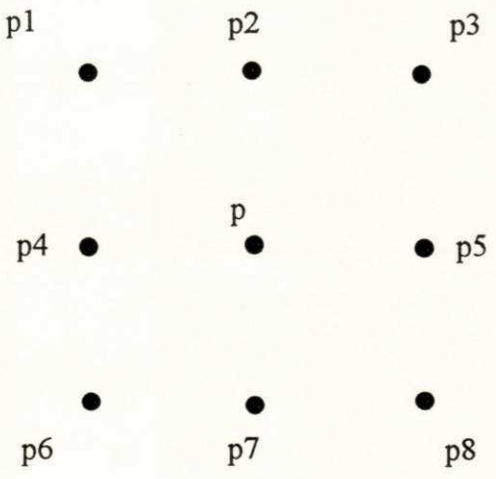
[8]Todos os métodos trabalham calculando o valor da função iluminação para cada ponto da imagem. O que diferencia um método do outro é a forma como, a partir da imagem, calculamos o vetor normal.

6.4.2.1) Malha triangular

Este método é uma proposta original da tese e consiste em determinar uma pequena malha triangular na vizinhança de cada ponto da imagem, e estimar o vetor normal no ponto a partir das normais na malha triangular.

A malha triangular é determinada da seguinte forma:

a) para cada ponto da imagem, encontramos seus 8 vizinhos (Figura 6. 7)



vizinhos

(i-1, j-1, k1)	(i-1, j, k2)	(i-1, j+1, k3)
(i, j-1, k4)	**(i, j, k)**	(i, j+1, k5)
(i+1, j-1, k6)	(i+1, j, k7)	(1+1, J+1, k8)

Figura 6. 7 - Oito vizinhos

b) para cada vizinho P_n, calculamos o vetor $\bar{v}_n$ através da diferença $P_n - P$ com $(1 \leq n \leq 8)$ (Figura 6. 8)

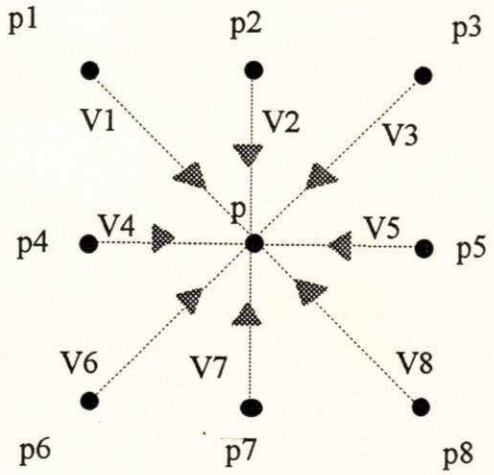


Figura 6. 8 - Vetores Pn

c) calculamos a seguir os vetores normais $\bar{N}_n$ através do produto vetorial $\bar{v}_n \wedge \bar{v}_{n+1}$ com $(1 \leq n \leq 7)$ (Figura 6. 9)

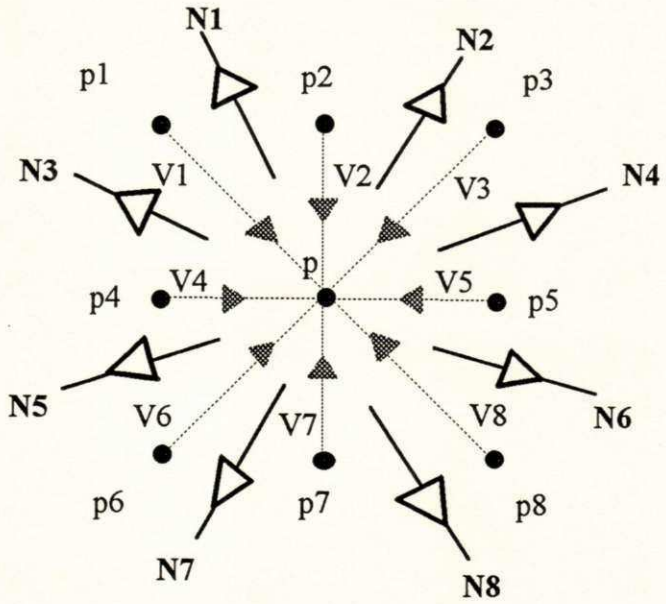


Figura 6. 9 - Vetores Nn

d) calculamos o vetor normal resultante $\bar{N}$ no ponto, através da soma dos vetores $\bar{N}_n$ (Figura 6. 10)

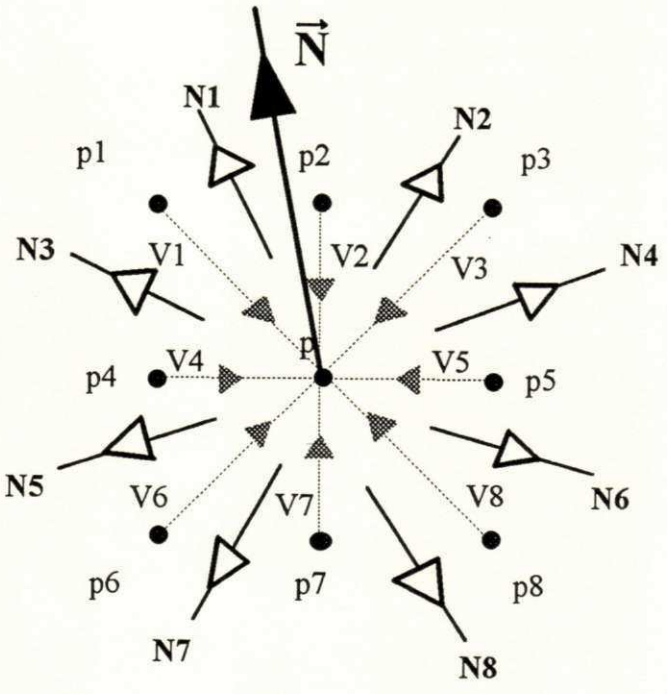


Figura 6. 10 - Vetor normal resultante

e) calculamos o vetor unitário $\bar{N}' = \bar{N} / \|\bar{N}\|$

Como pode ter sido observado, para os pontos das bordas da imagem, não temos os 8 vizinhos definidos. Neste trabalho optamos por desprezar todos este pixels de borda.

6.4.2.2) Gradient shading

Este método encontra-se descrito em [Chen85], [Gord85], [Hoeh86]. O método consiste em estimar a normal para cada ponto da superfície através do vetor gradiente para este ponto.

$$\nabla z = (\partial z / \partial x,\ \partial z / \partial y,\ 1)$$

Assim, existe a necessidade de obter as derivadas parciais : ($\partial z / \partial x$, $\partial z / \partial y$), numericamente.

A derivada " $\partial z / \partial x$ " pode ser obtida a partir de uma das diferenças :

. "backward difference" - $\delta bx = z_{i,j} - z_{i-1,j}$

. "forward difference" - $\delta f\, x = z_{i+1,j} - z_{i,j}$

. "central difference" - $\delta c\, x = \frac{1}{2} (z_{i+1,j} - z_{i-1,j})$

Do mesmo modo " $\partial z / \partial y$ " pode ser obtida de :

. "backward difference" - $\delta by = z_{i,j} - z_{i,j-1}$

. "forward difference" - $\delta f\, y = z_{i,j+1} - z_{i,j}$

. "central difference" - $\delta c\, y = \frac{1}{2} (z_{i,j+1} - z_{i,j-1})$

Das 3 formas de cálculo, segundo [Gord85] a "central difference" apresenta uma melhor aproximação desde que a imagem não seja descontínua, no entanto esta não é uma situação usual. (Figura 6. 11)

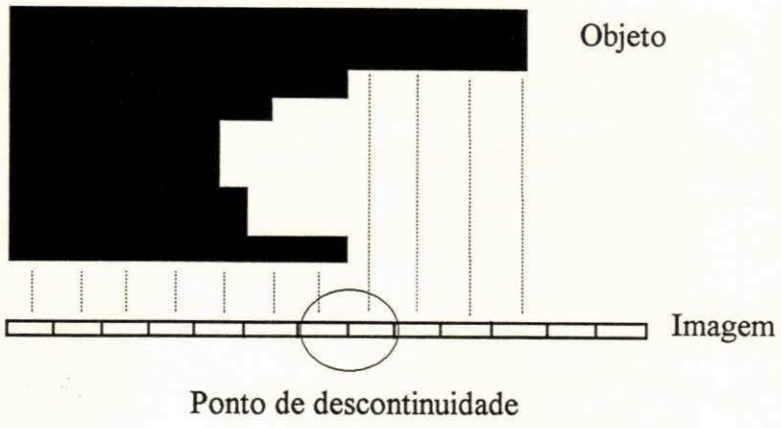


Figura 6. 11 - Descontinuidade simples na imagem

[Gord85] sugere como solução, adotar uma média ponderada entre δ_b e δ_f, com a seguinte função para cálculo dos pesos:

$W(\delta) = 1$, se $\delta \leq a$

ε^9, se $\delta \geq b$

$((1 + \varepsilon) / 2 + (1 - \varepsilon) / 2) \cos ((\delta - a) / (\delta - b) \pi)$, caso contrário

assim :

$$\partial z / \partial x = (W_{bx} \delta_{bx} + W_{fx} \delta_{fx}) / (W_{bx} + W_{fx})$$

$$\partial z / \partial y = (W_{by} \delta_{by} + W_{fy} \delta_{fy}) / (W_{by} + W_{fy})$$

A função "peso" procura basicamente atribuir um valor máximo de peso (1) para pequenos valores de δ , menores que um limite "a".

---

[9] ε é o menor número que dividido por 2ε mantém uma precisão considerada suficiente.

Para valores muito grandes de δ , maiores do que "b", ela atribui um peso mínimo.

Para valores de δ no intervalo [a, b] ela procura estabelecer pesos que decrescem continuamente, na medida em que δ cresce[10].

Analisando as situações possíveis temos :

Para $\delta_b = \delta_f$, nós temos o valor da "central difference";

para δ_b e δ_f , ambos com valores grandes ou pequenos, a função procura estabelecer um valor ponderado entre eles;

para δ_b ou δ_f quando um deles possuir um valor grande e o outro pequeno, a função prioriza o menor valor, por ser este considerado o mais provável para representar o verdadeiro declive da superfície.

segundo [Gord85] este método, para superfícies que possuam declives contínuos (constantes) e se projetem pelo menos em uma área de 3 x 3 pixels, consegue calcular o vetor normal correto.

O grande problema para os algoritmos que trabalham no espaço da imagem, se dá com relação aos pontos de descontinuidade dentro da imagem (Figura 6. 12), que podem ser minimizados mas não resolvidos completamente, isto porque as informações da superfície original já não estão mais disponíveis na imagem[11].

---

[10]Nos artigos de [chen85] e [gord85] foram utilizados :

$\varepsilon = 10^{-5}$

$b = 5$

$a = 2$

[11]O problema poderia ser minimizado ainda mais se produzissemos uma pré-imagem com resolução maior.

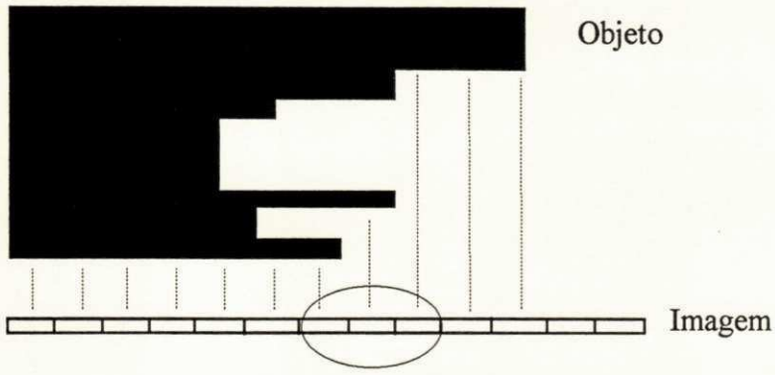


Figura 6. 12 - Descontinuidade complexa

CAPÍTULO 7

RECORTE

7.1) Introdução

Existem muitas operações que o usuário pode realizar sobre o modelo do objeto ou a imagem produzida do mesmo. Neste capítulo trataremos de uma operação aqui denominada de "recorte", através da qual é permitido ao usuário retirar partes do modelo.

Este tipo de operação pode ajudar ao usuário a observar detalhes internos do objeto modelado, que de outra forma não poderiam ser observados.

No capítulo de descrição do sistema, já foi detalhada a forma como o usuário interage para realizar esta operação de recorte. Assim, neste capítulo nos limitaremos a forma como o algoritmo de recorte trabalha.

7.2) Algoritmo

```
Início
    Para todo octante na octree faça :
    Início
        Chamar procedimento de recorte para o octante;
    Fim
Fim
```

```
Procedimento de recorte
Início
    Testar a posição do octante em relação ao plano de recorte;
    Se octante acima do plano de recorte e subespaço que deverá
    manter a porção restante do objeto é este faça :
        Armazenar este octante na nova octree;
    Se não, Se octante abaixo do plano de recorte e subespaço que
    deverá manter a porção restante do objeto é este faça :
        Armazenar este octante na nova octree;
    Se não, Se octante é interceptado pelo plano de recorte e octante
    não possui tamanho mínimo faça :
        Para cada um dos oito filhos deste octante faça :
            Chamar o procedimento de recorte para o filho;
Fim
```

7.3) Explicação do algoritmo

Podemos notar que este algoritmo independe da ordenação dos octantes na octree, e por isto podemos percorrer a octree seqüencialmente para recortar o objeto, o que diminui o tempo necessário para realizar o recorte.

Esta independência se deve ao fato de que o algoritmo trata cada octante individualmente, sem o relacionar com os outros octantes na octree.

O teste da posição do octante, em relação ao plano de recorte, pode ser realizado através de um teste dos oito vértices do octante, contra o plano de recorte.

Assim, se pelo menos um vértice se coloca em um lado oposto do plano de recorte, em relação aos outros vértices, então existe a intercessão do octante com o plano. Caso contrário, o octante se encontra totalmente de um dos lados do plano de recorte (Figura 7. 1).

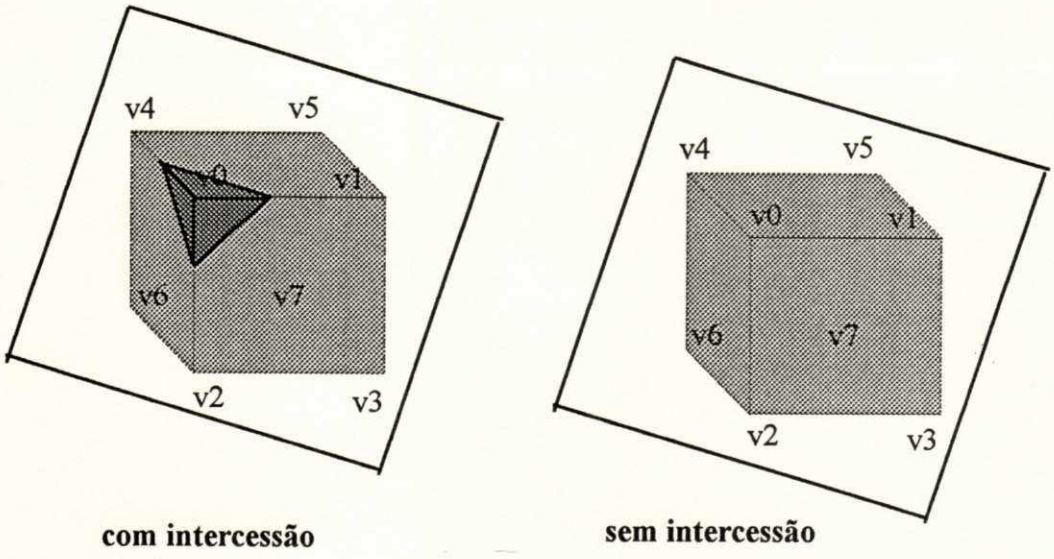


Figura 7. 1 - Teste de intercessão

A subdivisão de um octante[1] em seus oito octantes filhos é realizada da mesma forma como foi feita no algoritmo de projeção através de subdivisões recursivas. Assim, não entraremos em muitos detalhes, bastando apenas dizer que para realizá-la devemos simplesmente substituir o dígito menos significativo no código, que indique uma compactação, pelo número do octante a ser codificado (Figura 7. 2).

[1]Este octante não pode ter tamanho mínimo, porque assim ele não poderia ser subdividido.

Pai	Filhos
	12320X
	12321X
	12322X
1232X	12323X
	12324X
	12325X
	12326X
	12327X

Figura 7. 2 - Subdivisão de um octante

Podemos perceber também que os octantes de tamanho mínimo, que são interceptados pelo plano de recorte, foram removidos da octree resultante, o que é eficiente e razoável, dado que no caso dos octantes de tamanho mínimo, o teste de intercessão com o plano de recorte é realizado somente com o ponto central do octante (Figura 7. 3).

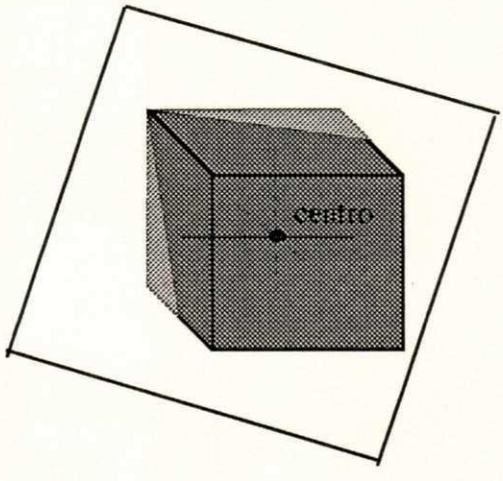


Figura 7. 3 - Teste de intercessão para um octante de tamanho mínimo

Assim, caso exista a intercessão sabemos que boa parte do octante está do outro lado do plano de recorte, justificando a opção de remoção.

CAPÍTULO 8

CONCLUSÃO

8.1) Introdução

O presente capítulo se dedica à apresentação, da metodologia empregada na verificação do funcionamento dos algoritmos implementados, dos resultados obtidos e finalmente da discussão desses resultados, propostas de melhorias e trabalhos futuros.

8.2) Teste dos algoritmos na reconstrução de objetos

Para a verificação foram utilizados dois tipos de objetos :

. matematicamente definidos - semiesfera, calota esférica, hiperbolóide de uma folha e a união da calota com o hiperbolóide.

. reais - escolhemos dois objetos, um pimentão e um crânio, por estes possuírem formatos geométricos complexos;

8.2.1) Descrição dos objetos e métodos de obtenção dos cortes

Para todos os objetos matemáticos foi utilizada a mesma técnica de varredura do espaço, na obtenção dos dados relativos aos cortes (slices) dos objetos.

A varredura consiste em percorrer uma parte do espaço (que esteja ocupada pelo objeto), delimitada pelos planos : x_i, x_f, y_i, y_f, z_i, z_f; variando-se sempre um valor fixo de deslocamento em cada direção (d_X, d_Y, d_Z) segundo o algoritmo abaixo :

```
Início
    Para y de yi até yf, incrementando de dy faça :
        Para x de xi até xf, incrementando de dx faça :
            Para z de zi até zf, incrementando de dz faça :
            início
                Aplicar os valores de (x, y, z) às equações que
                delimitam o objeto e verificar se o ponto testado
                é interno ou externo ao objeto
            Fim
Fim
```

O deslocamento na direção y (d_Y) foi calculado para fornecer sempre 60 cortes do objeto. Os deslocamentos d_X, d_Z são sempre iguais e permitem a obtenção dos cortes na resolução de 128 x 128 pontos.

Valores :

. planos delimitadores :

	inicial	final
X	25,2	-25,2
Y	0,0	20,0
Z	25,2	-25,2

. deslocamentos :

	deslocamento
X	- 0,4
Y	+ 0,3
Z	- 0,4

. equações :

semiesfera :
$r^2 = x^2 + y^2 + z^2$, com : $r^2 \leq 900$

calota esférica :
$r^2 = x^2 + y^2 + z^2$, com : $824 \leq r^2 \leq 900$

hiperbolóide de uma folha :
$p = (x^2 / 6^2) + (y^2 / 6^2) - (z^2 / 8^2)$, com $-1 \leq p \leq 1$

. Pimentão

Um pimentão de 12 cm de comprimento foi marcado com 3 eixos (com aproximadamente 120^o entre eles), preenchido com gelatina na concentração de 0,5 molar e congelado. Após o endurecimento da gelatina, este foi cortado em 23 fatias de 0,5 cm de espessura aproximadamente. Em seguida, cada uma destas fatias foi lavada em água morna (para retirar a gelatina) e digitalizada na resolução de 128 x 128 pixels.

Erros associados ao processo :

. dificuldade na marcação correta dos eixos de alinhamento e conseqüentemente, erros no alinhamento das fatias;

. dificuldade em se obterem cortes com a mesma espessura, que após digitalizados foram considerados iguais;

. dificuldade em se obterem cortes paralelos, que após a digitalização foram considerados paralelos;

. número pequeno de fatias, dada a dificuldade de obtenção de fatias finas, implicando em um aumento no erro associado ao volume encontrado entre dois cortes.

. Crânio

Neste caso, foram obtidos 33 tomogramas da região superior de um crânio humano, com a distância entre cortes igual a 0,3 cm. Os tomogramas foram gerados inicialmente em filme e por isto tiveram que sofrer novo processo de digitalização, na resolução de 128 x 128 pixels.

Erros associados ao processo :

. dificuldade em realizar um alinhamento regular dos tomogramas;

. necessidade de redigitalir as imagens dos tomogramas obtidos;

. número pequeno de tomogramas do exame obtido.

8.2.2) Resultados visuais

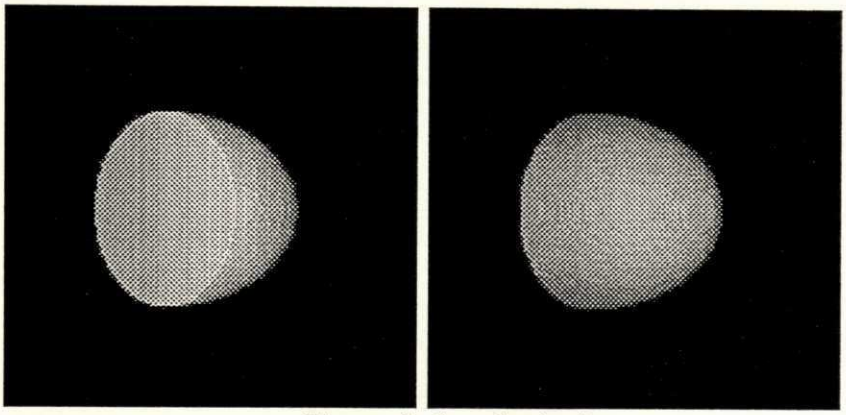


Figura 8. 4 - Semiesfera

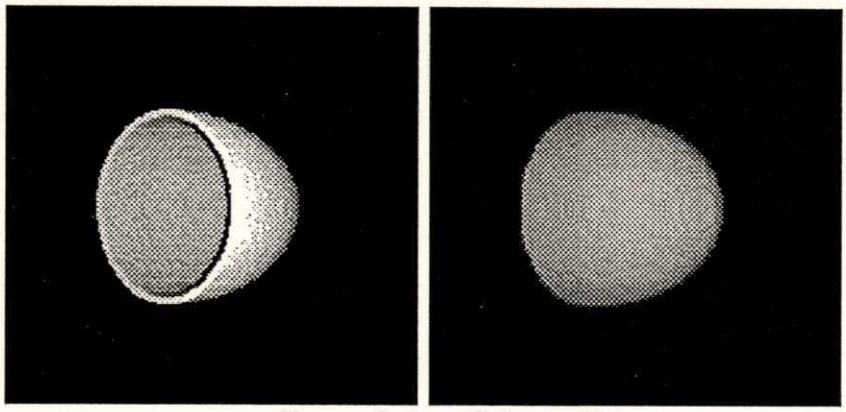


Figura 8. 5 - Calota esférica

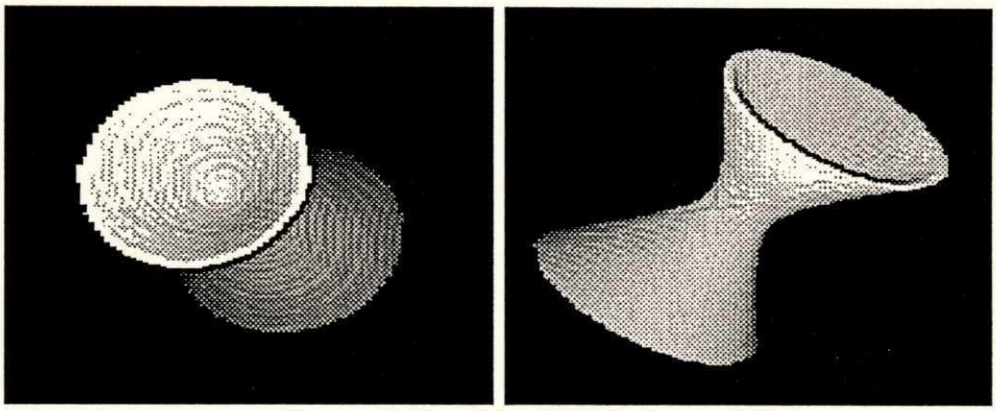


Figura 8. 6 - Hiperbolóide de uma folha

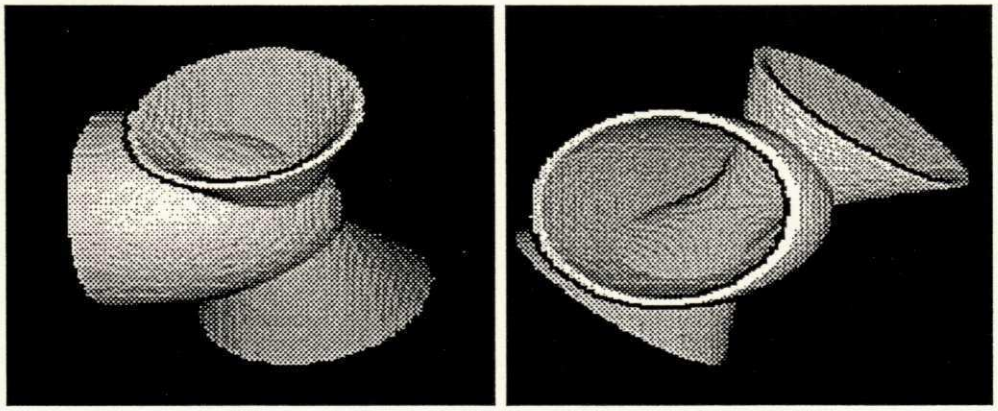
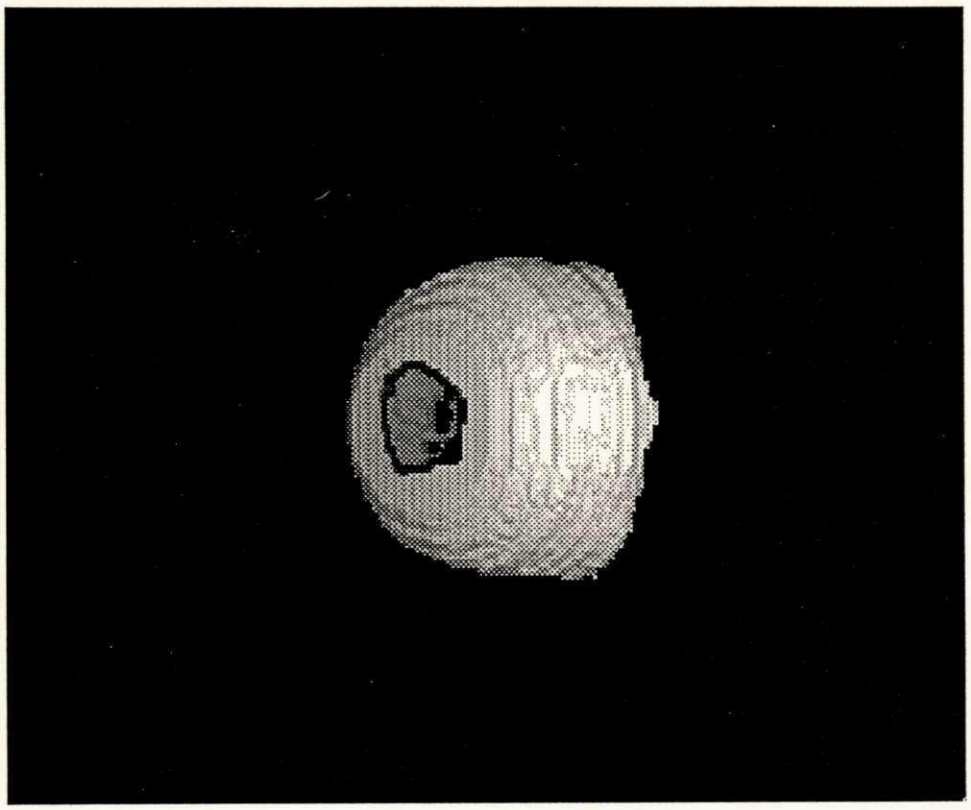


Figura 8. 7 - União da calota com o hiperbolóide



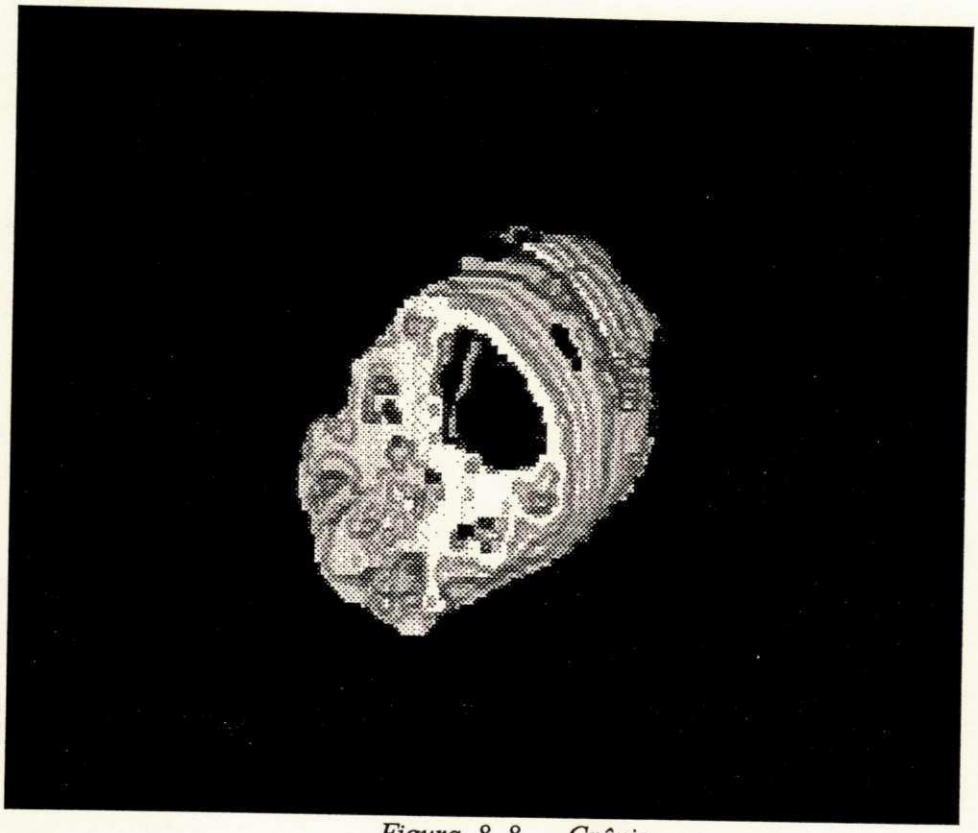
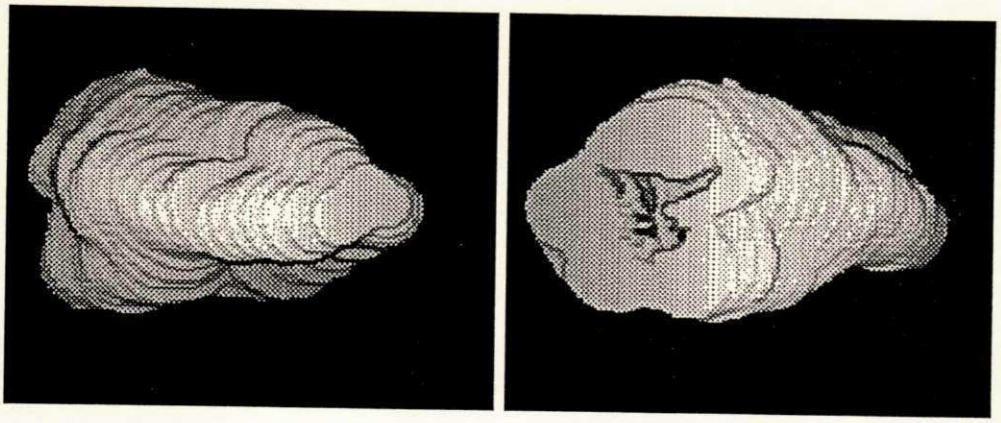


Figura 8. 8 - Crânio



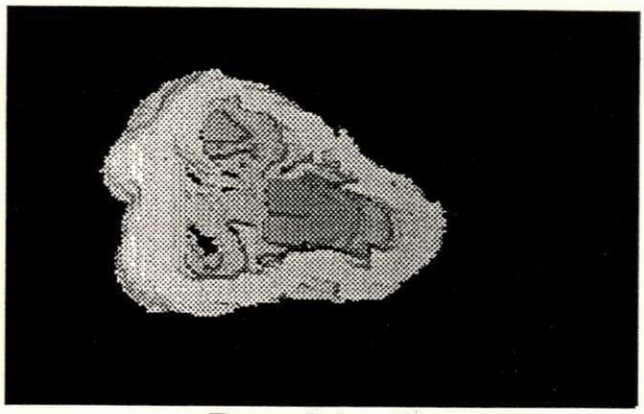


Figura 8. 9 - Pimentão

8.2.3) Resultados analíticos

8.2.3.1) Modelagem

Mostraremos a seguir, uma tabela que compara o número máximo de elementos dos modelos octree e cuberille para algumas resoluções e explicaremos como esta foi obtida :

número de subdivisões	total de divisões por eixo	total de elementos na cuberille	número máximo de elementos na octree linear
0	2^0 1	$(2^0)^3$ 1	1
1	2^1 2	$(2^1)^3$ 8	$7x(8)^0$ 7
2	2^2 4	$(2^2)^3$ 64	$7x(8)^1$ 56
3	2^3 8	$(2^3)^3$ 512	$7x(8)^2$ 448
4	2^4 16	$(2^4)^3$ 4096	$7x(8)^3$ 3584
5	2^5 32	$(2^5)^3$ 32768	$7x(8)^4$ 28672
6	2^6 64	$(2^6)^3$ 262144	$7x(8)^5$ 229376
7	2^7 128	$(2^7)^3$ 2097152	$7x(8)^6$ 1835008
8	2^8 256	$(2^8)^3$ 16777216	$7x(8)^7$ 14680064

O número máximo de elementos na octree linear foi calculado supondo o penúltimo nível da octree preenchido somente com sete octantes de tamanho mínimo, o que não permite a junção de nenhum grupo de oito elementos garantindo a existência de todos os octantes em seu tamanho mínimo (Figura 8. 1).

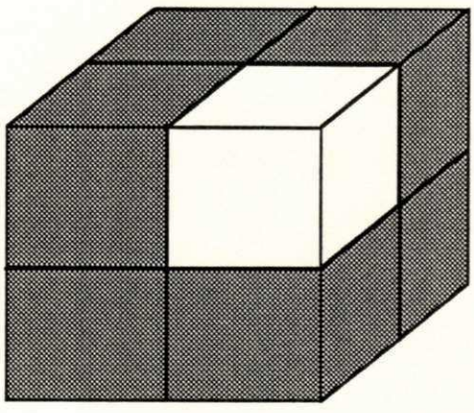


Figura 8. 1 - Penúltimo nível da octree

Assim, como temos o total de octantes de tamanho mínimo igual a (8^{n}), onde "n" é o número de subdivisões da octree, temos o número máximo de octantes igual a ($7 \times 8^{n-1}$) elementos.

Agora que sabemos o número máximo de octantes em uma octree linear, podemos calcular o tamanho máximo (em bytes) da octree, o que permitirá uma análise da economia de memória obtida com esta modelagem.

número de subdivisões	total de bits por código	total de bytes por código	tamanho máximo da octree linear	tamanho máximo da cuberille
0	1 4	1	1	1
1	1 x 4 4	1	7	1
2	2 x 4 8	1	56	8
3	3 x 4 12	2	896	64
4	4 x 4 16	2	7168	512
5	5 x 4 20	3	86016	4096
6	6 x 4 24	3	688128	32768
7	7 x 4 28	4	7340032	262144
8	8 x 4 32	4	58720256	2097152

Podemos notar que o tamanho máximo da octree linear é (em média) 20 vezes maior do que o tamanho do modelo cuberille[1] equivalente[2].

Na prática este tamanho máximo raramente ocorre, pois os objetos possuem uma característica chamada ***coerência espacial*** que permite a obtenção de octrees bem menores.

Para que se tenha uma idéia desta diminuição foi realizada uma simulação com dois objetos : uma calota esférica que possui baixa coerência espacial e a mesma calota preenchida (o que aumenta a sua coerência espacial) produzindo uma semiesfera. Os dados deste teste foram colocados na tabela abaixo[3] :

---

[1]Estamos supondo uma implementação muito econômica de uma cuberille, onde cada elemento desta ocuparia um bit em uma matriz cúbica de 2n x 2n x 2n bits.

[2]O modelo cuberille possui tamanho fixo para uma dada resolução.

[3]Colocamos também o tamanho do modelo cuberille para o mesmo objeto, e o percentual de economia da octree em relação a cuberille.

	número de octantes	tamanho da octree em bytes	tamanho da cuberille em bytes	percentual de redução
calota	31041	124164	262144	52,64%
semiesfera	18420	73680	262144	71,90%

Assim, podemos perceber que, quanto maior a coerência espacial do objeto, mais adequado ele é para ser modelado por uma octree.

Para finalizar este tópico, mostramos abaixo o tempo gasto na modelagem dos dois objetos da simulação.

	número de octantes	tempo de modelagem
calota	31041	64 s
semiesfera	18420	185 s

Como se pode notar, a octree para a semiesfera é menor mas possui um tempo de modelagem 3 vezes maior do que o tempo gasto para construir a octree para a calota esférica.

Este tempo gasto a mais é devido ao tempo de ordenação e "compactação" dos octantes, para produzir uma octree que modela um objeto de volume maior mas possui um modelo menor.

8.2.3.2) Projeção

Como foi explicado no capítulo sobre projeção, neste trabalho foram estudados três algoritmos para realizar a projeção dos modelos octree. Aqui compararemos somente os dois algoritmos que apresentaram melhor desempenho em termos de tempo, "perfuração direta" e "z buffer".

Visando dar seqüência à discussão, apresentaremos em seguida uma tabela com os tempos de projeção dos dois algoritmos, utilizando os dois objetos da simulação em uma resolução de (320 x 200) com ângulos diferentes :

	perfuração direta		**z buffer**	
azimute / elevação	45° / 45°	0° / 45°	45° / 45°	0° / 45°
calota	456 s	376 s	14 s	14 s
semiesfera	387 s	379 s	50 s	50 s

Podemos notar na tabela anterior que o algoritmo de z buffer (que apresentou um melhor desempenho) possui tempo de projeção dependente somente do objeto[4] , enquanto o algoritmo de perfuração direta possui tempo de projeção dependente tanto do objeto projetado quanto de sua orientação no espaço. Isto de deve basicamente a dois fatores :

---

[4]No que diz respeito ao volume ocupado pelo objeto, isto porque um objeto com volume maior irá produzir um número maior de subdivisões, acarretando um aumento no tempo de projeção.

a) dependendo do formato da superfície do objeto, o algoritmo de perfuração direta leva mais tempo para percorrer o modelo e encontrar a intercessão (Figura 8. 2);

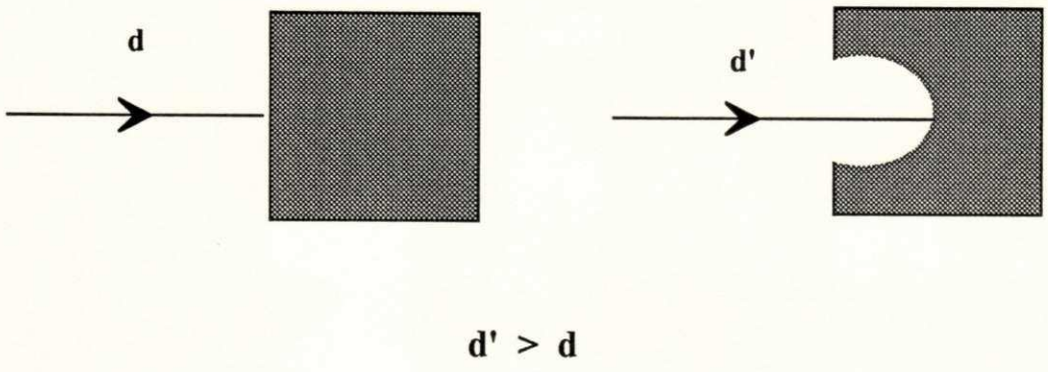


Figura 8. 2 - Superfícies com formatos diferentes

b) dependendo da orientação do objeto no espaço (em relação ao plano de projeção), existe maior porção de espaços vazios na projeção, o que faz com que o raio traçado tenha que percorrer todo o espaço do universo sem encontrar a intercessão (Figura 8. 3).

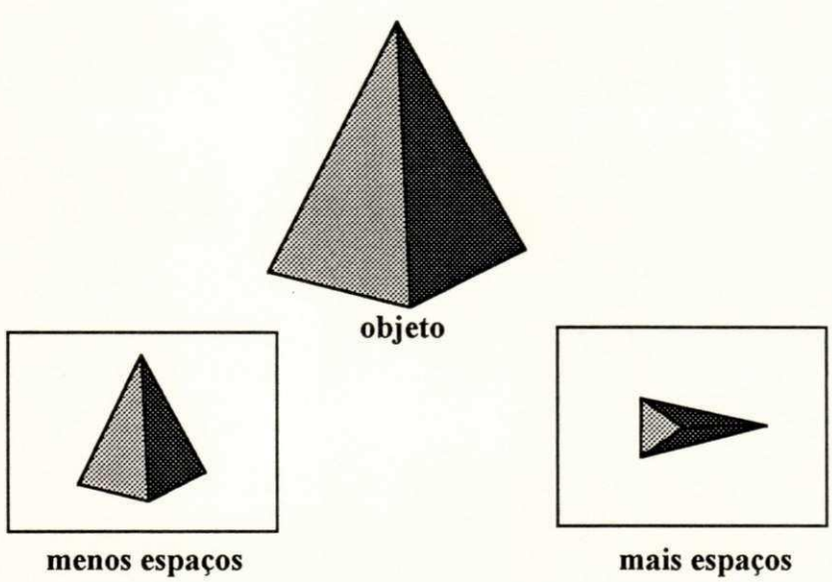


Figura 8. 3 - Espaços vazios nas projeções

No que diz respeito a qualidade da projeção, não foi sentida nenhuma diferença entre os algoritmos[5] .

Isto talvez se deva ao fato de estarmos trabalhando com uma imagem de baixa resolução (max : 320 x 200), devido às limitações da placa de vídeo VGA no modo de 256 cores simultâneas.

8.2.3.3) Iluminação

No tocante à iluminação, não foi nosso objetivo propor novas técnicas para a solução desta questão. Nos limitamos, outrossim à implementação de modelos conhecidos e fartamente publicados, com a única exceção no método de "malha triangular"[6]. A intenção aqui era simplesmente colocar à disposição do usuário esta facilidade, que é fundamental na produção de imagens de boa qualidade e na visualização de pequenos detalhes da imagem (Figura 8. 4).

---

[5]Não foi estabelecida nenhuma métrica para avaliar a qualidade da projeção, a avaliação foi simplesmente visual. Como esta avaliação é muito imprecisa, sugerimos a leitura do artigo de [hoeh86] que possui algumas sugestões de como medir esta qualidade.

[6]Ver capítulo 6

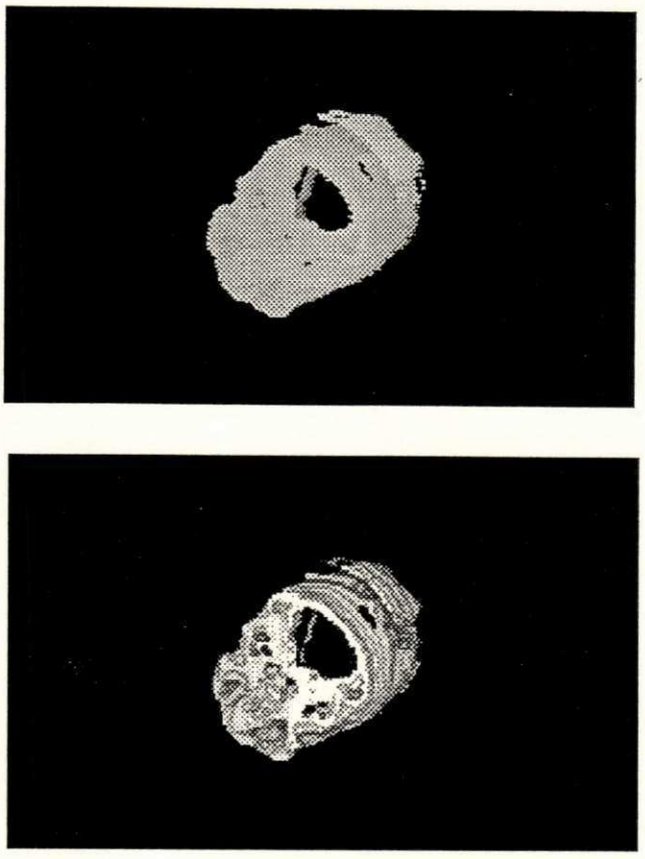


Figura 8. 4 - Crânio com iluminação : somente pela distância e pelo modelo completo

No entanto implementamos dois métodos de iluminação e dois modelos de iluminação, na intenção de poder realizar pequenas comparações que discutiremos agora.

Em termos de tempo, as diferenças tanto no método como no modelo, para a resolução trabalhada, foram mínimas como se pode notar na tabela abaixo :

método / modelo	malha / completo	malha / difuso	vizinho / completo	vizinho / difuso
calota	6 s	4 s	4 s	3 s
semiesfera	6 s	4 s	4s	3 s

Em termos de qualidade no cálculo do vetor normal, o método da malha triangular mostrou ser superior (Figura 8. 5), com a vantagem adicional (para o usuário) de não ser necessário informar nenhum parâmetro para o método.

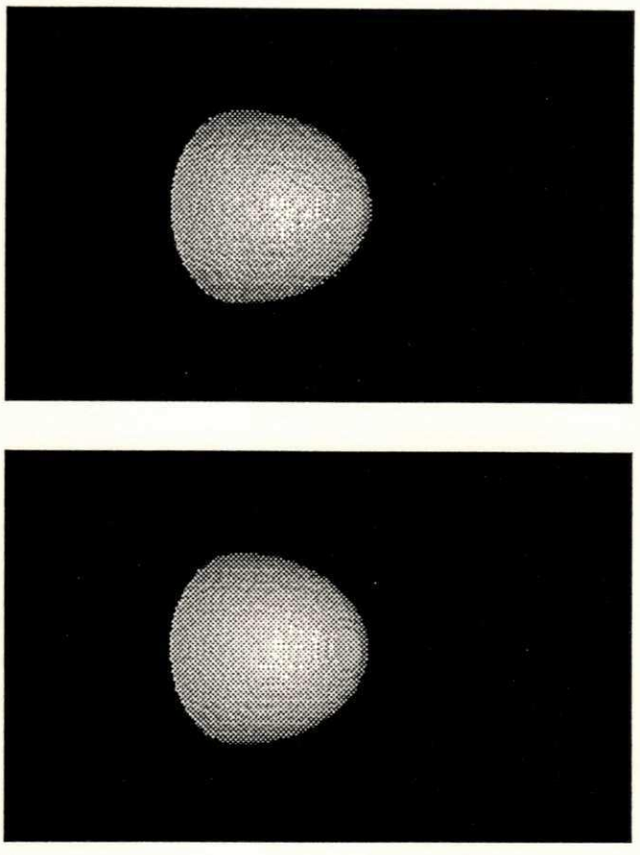


Figura 8. 5 - Vetor normal (vizinho e malha)

Em termos do problema de descontinuidade na imagem, que é a principal dificuldade a ser tratada quando se esta trabalhando no espaço da imagem (ver capítulo sobre iluminação), este apareceu em ambos os métodos (malha, vizinho), sendo que o problema se mostrou um pouco mais acentuado quando utilizamos o método de malha triangular (Figura 8. 6).

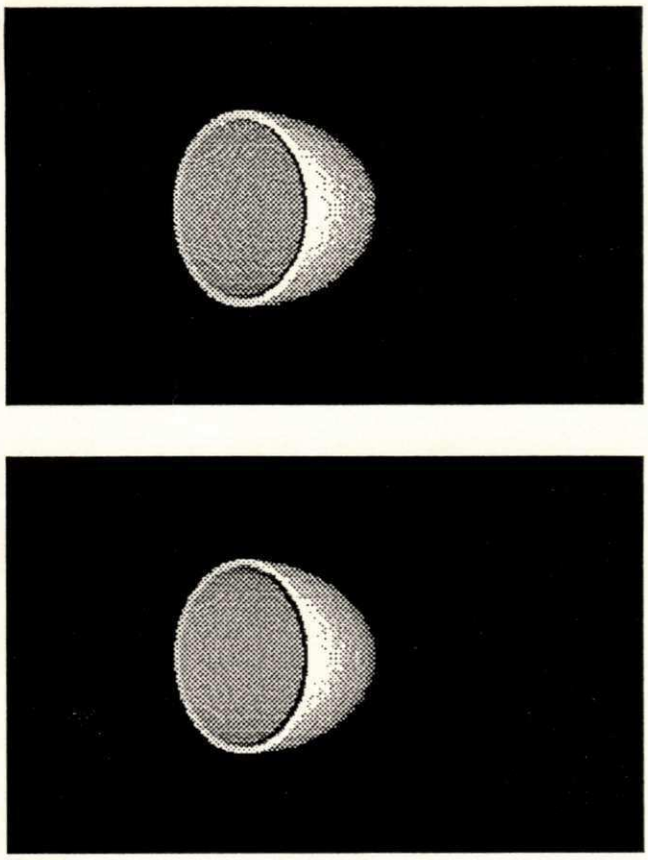


Figura 8. 6 - Descontinuidade (malha e vizinho)

No que diz respeito ao modelo de iluminação, ambos apresentaram bons resultados para a resolução trabalhada, sendo que o modelo de iluminação completo permite um controle mais "fino" do efeito luminoso, mas com a desvantagem de possuir um ajuste mais trabalhoso dado o grande número de parâmetros que se deve informar.

8.3) Limitações impostas ao trabalho

Existiram muitas dificuldades de ordem pessoal e material que implicaram em limitações à qualidade dos resultados obtidos. Agora citaremos algumas dessas dificuldades :

. impossibilidade de obtenção dos tomogramas em meio magnético, devido ao desconhecimento do formato de gravação destas imagens;

. dificuldade na obtenção de uma quantidade adequada de tomogramas, retirados com espaçamento constante entre eles, devido ao alto custo de um exame deste tipo;

. indisponibilidade de : controladora, monitor de vídeo e driver para uma resolução super VGA, o que permitiria a obtenção de imagens em uma resolução maior que a atual (320 x 200 pixels);

. limitação na quantidade de tons de uma cor primária (32 tons), o que não permitiu a obtenção de uma palette com tons de cinza linear e uniforme;

8.4) Trabalhos futuros

No que diz respeito a possíveis trabalhos, que possam utilizar o trabalho atual como base, não é possível prever todos eles.

Entretanto, nos artigos consultados, existe grande propensão de aplicar este tipo de modelagem para a área médica, principalmente no que diz respeito a projetos de cirurgias, planos de terapias com radiação e elaboração de próteses ortopédicas.

8.5) Melhorias ao trabalho atual

Existem muitos pontos que podem ser melhorados neste trabalho, acreditamos que os principais são :

a) elaboração de métricas para avaliar a qualidade da modelagem, projeção e iluminação;

b) permitir a visualização de mais de uma octree simultaneamente;

c) implementar melhores algoritmos de classificação para as imagens;

d) avaliar e implementar, caso seja necessário, a interpolação linear entre cortes para diminuir possíveis erros na modelagem;

e) implementar otimizações do tipo "back to front", para o algoritmo de z buffer, melhorando ainda mais o seu desempenho;

f) investir no estudo de heurísticas que resolvam o problema de se traçar raios que não irão tocar o objeto;

g) estudar e implementar alguma estratégia que resolva o problema do cálculo do vetor gradiente em pontos descontínuos da imagem.

BIBLIOGRAFIA

[Abra85] ABRAM, G. e L. Westover, Efficient Alias-free Rendering using Bit-mask and Look-up Tables, **ACM SIGGRAPH**, San Francisco, july 1985, pag. 53-59

[Aman85] AMANS, Jean Louis e P. Darier, Processing and Display of Medical Three Dimensinal Arrays of Numerical Data Using Octree Encoding, **Medical Image Processing SPIE** 593, 1985, pag. 78-88

[Berg86] BERGMAN, L. , H. Fuchs e E. Grant, Image Rendering by Adaptive Refinement, **ACM SIGGRAPH**, Dallas, august 1986, pag. 29-37

[Bold84] BOLDRINI, J. L. e Sueli I. Rodirgues, **Álgebra Linear**, Harper & Row do Brasil - São paulo, 1984

[Boma90] BOMANS, Michael , Karl-Hienz Höhne, 3-D Segmentation of MR Images of the Head for 3-D Display, **IEEE Transaction on Medical Imaging**, v(9), n(2), june 1990, pag. 177-183

[Boul87] BOULOS, P. e Ivan de Camargo, **Geometria Analítica : Um Tratamento Vetorial**, Mc Graw-Hill - São Paulo, 1978

[Buit75] BUIT-TUONG, Phong, Illumination for Computer Generated Pictures, **CACM,** v(18), n(6), June 1975, pag. 311-317

[Carl85] CARLBOM, I. , I. Chakravarty e A Hierarchical, Data structure For Representing The Spatial Decomposition of 3D-Objects, **IEEE Computer Graphics and Applications**, april 1985, pag. 24-31

[Catm74] CATMULL, E. , **A Subdivision Algorithm for Computer Display of Curved Surfaces,** Ph. D. Thesis, Report UTEC - Cse - 74 - 133, Computer Science Department, University of Utah, Salt Lake City, UT, december, 1974

[Chen85] CHEN, L. S. , G. T. Herman e R. A. Reynolds, Surface Shanding in the Cubirille Enviroment, **IEEE Computer Graphis and Image Processing**, December 1985, pag. 33-43

[Chui91] CHUI, Paul, A C++ PCX File Viewer for Windows 3, **Dr. Dobb's Journal**, july 1991, pag. 62-102

[Cline88] CLINE, H. E. e W. E. Lorensen, Two Algorithms For The Three-Dimensional Reconstruction Of Tomograms, **Med. Phys** 15(3), may/jun 1988, pag. 320-327

[Cols76] COLSHER, J. G. , Iterative Three-Dimensinal Image Reconstruction from Tomographis Projections, **Computer graphics and image Processing**, june 1976, pag. 513-537

[Dreb88] DREBIN, R. A. , L. Crapenter e P. Hanraham, Volume Rendering, **Computer Graphics**, 22(4), august 1988, pag. 65-74

[Duff85] DUFF, T. , Compositing 3-D Rendered Images, **ACM SIGGRAPH**, San Francisco, july 1985, pag. 41-44

[Frid85] FRIEDER, G. , D. Gordon e R. A. reynolds, Back-to-Front Display of Voxel-Based Objects, **IEEE CG&A**, january 1985, pag. 52-60

[Fole82] FOLEY, J. D. e A. Van Dam, **Fundamentals of Interactive Computer Graphics**, Addison Wesley, 1982

[Garg81] GARGANTINI, Irene, Liner Octrees For Processing of Three-dimensinal Objects, **IEEE Computer Graphis and Image Processing** 20, september 1981, pag. 365-374

[Garr90] GARRITY, M. P. , Raytracing Irregular Volume Data, **Computer Graphics**, 24(5), november 1990, pag. 35-40

[Glas84] GLASSNER, Andrew S. , Space Subdivision for Fast Ray Tracing, **IEEE CG&A**, october 1984, pag. 15-22

[Glas90] GLASSNER, Andrew S. , Ray Tracing for Realism, **BYTE**, december 1990, pag. 263-270

[Gord85] GORDON, Dan e R. A. Reynolds, Image Space Shanding of 3- Dimensional Objects, **Computer Vision, Graphis and Image Processing** 29, 1985, pag. 361-376

[Hear86] HEARN, D. e M. Pauline Baker, **Computer Graphics**, Prentice Hall - New Jersey, 1986

[Herm87] HERMAN, G. T. , Display of 3D Medical Images, **Medical Imaging Reseach** 160, march 1987, pag. 15-23

[Hoeh86] HOEHME, K. H. e R. Bernstein, Shanding 3D-Images From CT Using Gray-Level Gradiens, **IEEE Transaction on Medical Imaging**, march 1986, pag. 45-47

[Hoeh87] HOEHME, K. H. , M. Rimer e U. Tiede, 3D-display of Spatial Image Sequences, **Medical Imaging Reseach** 160, march 1987, pag. 25-35

[Hong87] HONG, Tsai-Hong e M. O. Shneier, Rotation and Translation of Objects Repreented by Octrees, **Proc. IEEE International Conference On Robotics**, 1987, pag. 947-952

[Horn87] HORN, Berthold Klaus, **Robot Vision**, MIT Press, 1987

[Jack80] JACKINS, C. L. e S. L. Tanimoto, Oct-trees and their use in Representing three-dimensional Objcts, **Computer Graphis and Image Processing**, july 1980, pag. 249-270

[Kaji84] KAJIYA, James e B. P. V. Herzen, Ray Tracing Volume Densities, **Computer Graphics**, 18(3), july 1984, pag. 165-173

[Knut73] KNUTH, D. E. , **The Art of Computer Programming**, Addison Wesley - Massachusets, 1973

[Lore87] LORENSEN, W. E. e H. E. Cline, Marching Cubes: a High Resolution 3D Surface Construction Algorithm, **ACM Computer Graphics** 21(4), july 1987, pag. 163-169

[Leve68] LEVENS, A. S. e John Wiley, **Graphics : Analysis and Conceptual Design**, Inc. Sons, 1968

[Levo88] LEVOY, M. , Volume Rendering, **IEEE Computer Graphis and Applications**, may 1988, pag. 29-37

[Meag82] MEAGHER, D. , Geometric Modeling Using Octree Encoding, **Computer Graphis and Image Processing** 19, june 1982, pag. 129-147

[Meag85] MEAGHER, D. , The Application of Octree Thechniques to 3-D Medical Imaging, **IEEE Seventh Annual Conference of the Engineering in Medicine and Biology Society,** 1985, pag. 612-615

[Requ80] REQUICHA, A. A. G. , H. B. Voelcker, Solid Modeling, **IEEE CG&G**, october 1983, pag. 25-37

[Sabl88] SABELLA, P. , A Rendering Algorithm for Visualizing 3D Scalar Fields, **Computer Graphics**, 22(4), august 1988, pag. 51-58

[Scot87] SCOTT, W. W. , E. K. Fishman e D. Magid, Acetabular Fractures: Optimal Imaging, **Radiology**, november 1987, pag. 537-539

[Spee89] SPEES, B. , Accessing Extnded Memory, **The C Users Journal**, november 1989, pag. 125-130

[Stev87] STEVENS A. , **Memory-Resident Utilities Screen I/O and Programming Techniques**, Mis:Press, 1987

[Sutt90] SUTTY, G. e Steve Blair, **Advanced Programmer's Guide to SuperVGAs**, Brady Books - New York, 1990

[Tani74] TANIMOTO, S. , T. Pavlidis, A Hierarchical Data Structure for Picture Processing, **Computer Graphics and Image Processing**, v(4), 1975, pag. 104-119

[Tied90] TIEDE, U. , K. H. Hoehme, M. Bomans e A. Pommert, Invertigation of Medical 3D-Rendering Algoritms, **IEEE Computer Graphics and Applications,** march 1990, pag. 41-53

[Toen90] TOENNIES, K. D. e U. Tronnier, 3D Modeling Using An Extended Cell Enumeratio Representation, **Comuper Graphics**, 24(5), november 1990, pag. 13-20

[Torb81] TORBORG, John G. e Louis J. , Display Tecniques For Octree- Encoded Objects, **IEEE Computer Graphics and Applications**, july 1981, pag. 29-38

[Upso88] UPSON, C. e M. Keeler, V-BUFFER: Visible Volume Rendering, **ACM SIGGRAPH**, Atlanta, august 1988, pag. 59-64

[Weng85] WENG, J. e N. Ahuja, Octree Representation of Object In Arbitrary Motion, **Proc CVPR**, San Francisco, CA, 1985, pag. 524-529

[Whit80] WHITTED, T. , An Improved Illumination Model for Shaded Display, **CACM,** v(23) , n(6), 1980, pag. 343-349

[Wilh90] WILHELMS, J. e A. V. Gelder, Octrees For Fast Isosurface Generation Extended Abstract, **Computer Graphics** 24(4), nov 1990, pag. 57-62

[Will90] WILLIAMS, Al, DOS + 386 = 4 Gigabytes Directly Address 4 Gigabytes of Memory in DOS from Yoour C or Assembly Language Applications, **Dr. Dobb's Journal**, pag. 62-71

[Yama84] YAMAGUCHI, K. e K. Fujimura, Octree-Related Data Structures and Algoritms, **IEEE Computer Graphics and Applications**, jan. 1984, pag. 53- 59